

Microsoft[®] Language シリーズ

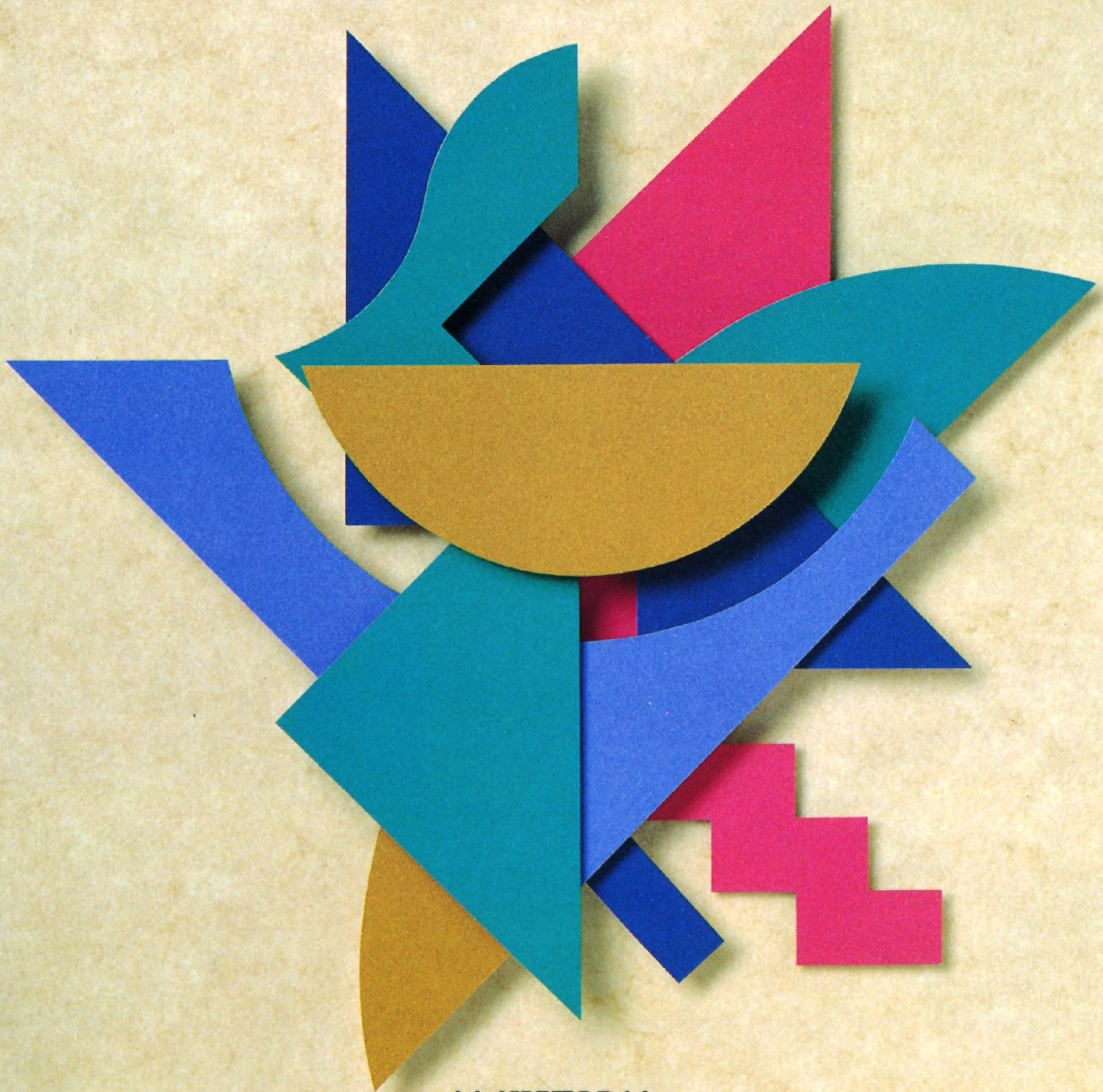
4

Ver. 4.2~4.5

Quick BASIC

初級プログラミング入門[上]

河西朝雄^[著]



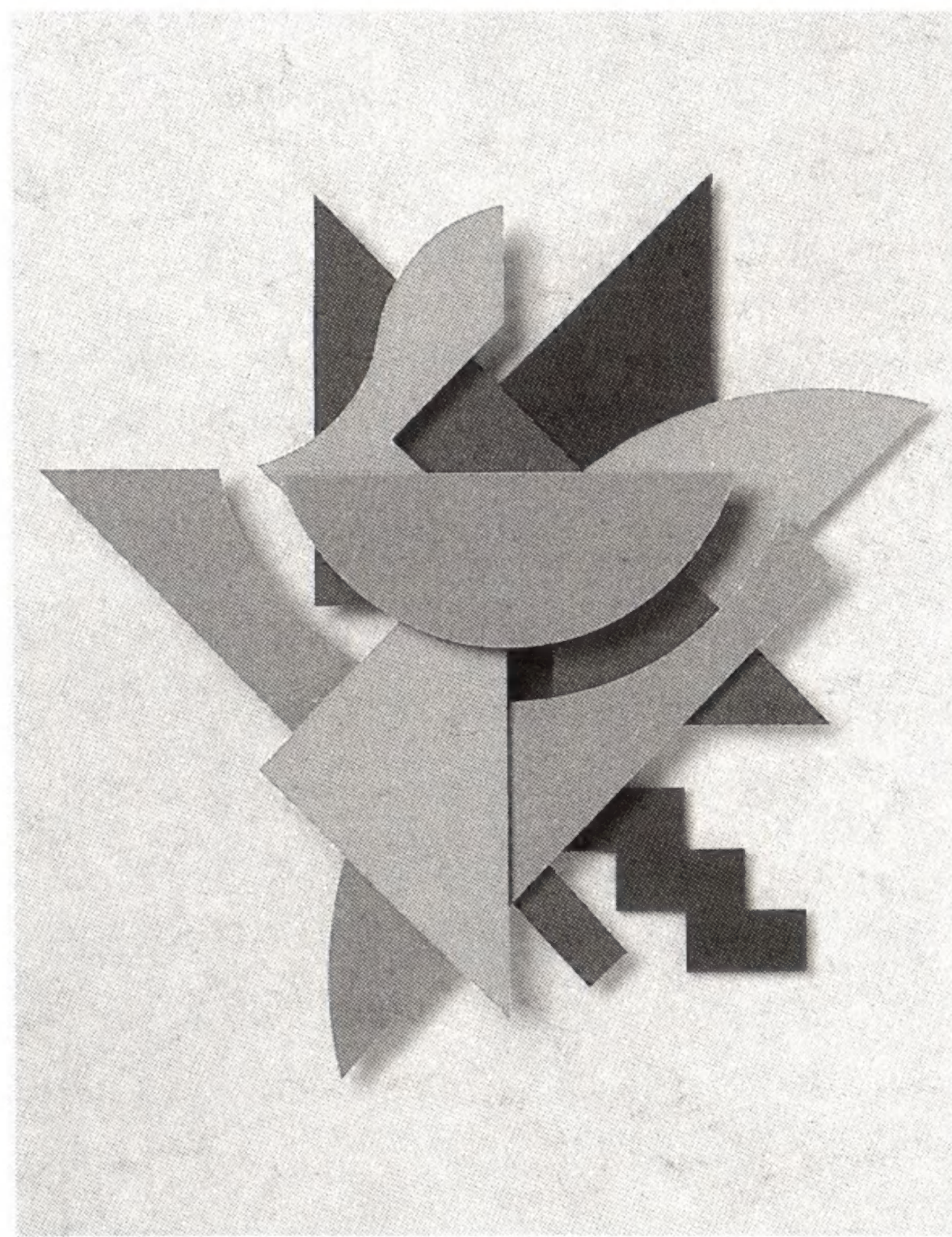
技術評論社

Ver.4.2~4.5

Quick BASIC

初級プログラミング入門[上]

河西朝雄^[著]



- ◎MS-DOS, QuickBASIC, QuickCは米国Microsoft社の登録商標です.
- ◎ATOK6は(株)ジャストシステムの登録商標です.
- ◎その他の商品は各社の登録商標, 商標または商品です. なお, 本書ではTM, ®の明記はしていません.

はじめに

従来の BASIC の大きな欠点は構造化プログラミングが行いにくいことです。しかし、BASIC の持つ使いやすさ、親しみやすさ、曖昧さなどのユーザ・フレンドリな良い環境を残しながら、BASIC に構造化という概念を導入すれば、そこに新しい BASIC の可能性が開けるのです。

このような観点から、Quick BASIC という処理系が Microsoft 社から発表されました。Quick BASIC は、

- 統合開発環境
- プログラムの構造化

という 2 つの強力な武器を備えた新世代言語です。

本書は Quick BASIC の操作法からプログラミング法を系統的にわかりやすくまとめたハンドブックで、次のような構成からなっています。

- 第 1 章 Quick BASIC の概要
- 第 2 章 統合開発環境 (QB)
- 第 3 章 コマンド・バージョン・ユーティリティ
- 第 4 章 Quick BASIC の言語仕様
- 第 5 章 ステートメントと関数の概要
- 第 6 章 ステートメント・関数リファレンス
- 第 7 章 Ver.4.2 から Ver.4.5 への変更点

国内での Quick BASIC は、Ver.4.2、Ver.4.5 と発展してきていますが、2 つのバージョンにおいて基本的には大きな違いはありません。そこで本書の本文は Ver.4.2 に基づいて解説し、Ver.4.2 と Ver.4.5 の違いを第 7 章で説明しました。

Quick BASIC は新しい BASIC の道を切り開いてゆく可能性を持った言語といえます。Quick BASIC を用いて、これから新しい BASIC の世界へ挑戦しようとしているユーザの方々にとって本書が、多少なりともお役に立てれば幸いです。

1990年 3 月 河西朝雄

目次

第1章 QuickBASIC の概要

1-1	QuickBASIC の特徴	12
1-2	システムの構築	14
1	提供されるソフトウェア	14
2	環境の設定	17
	■メモリの設定	17
	■パスの設定	18
	■日本語 FEP の組み込み	18
	■マウス・ドライバの組み込み	19
	■プリンタ・ドライバの組み込み	19
	■ライブラリの選択	20
3	システムの構築例	20
	■SETUP ユーティリティ	20
	■1MB フロッピーでの構築例	22
	■ハードディスクでの構築例	24
1-3	QuickBASIC の運用形態	26

第2章 統合開発環境(QB)

2-1	QB の起動とメニュー画面	28
	■QB の起動オプション	29
2-2	Quick メニューの使い方	30
1	メニュー操作の基本	30
	■メニュー操作の方式	30
	■メニュー操作で使うキー	31
	■マウス入力方式	31
	■サブメニューの意味	32
2	Quick メニュー一覧	33
3	ショート・カット・キー	36
2-3	ウィンドウ操作	37
1	ウィンドウの分割	37
2	アクティブ・ウィンドウの移動	38
3	ウィンドウの拡大縮小	38
4	実行画面への切り換え	38
2-4	ファイル操作	39
1	プログラムのセーブ	39
2	新規プログラムを作る	40

3	プログラムのロード	41
4	プログラムのプリントアウト	41
5	モジュール管理	42
	■サブファイルのロード	42
	■モジュール画面の切り換え表示	43
	■モジュールの解放	44
	■サブモジュールの作成	45
	■メインモジュールの設定	46
6	ドキュメント・ファイルの作成	47
7	インクルード・ファイル	48
	■インクルード・ファイルの作成	48
	■インクルード・ファイルの表示	48
	■インクルード・ファイルの編集	49
2-5	編集(エディタ)	50
1	キー配置	50
2	編集コマンド一覧	51
3	自動構文チェック機能	53
4	オート・インデント機能	54
5	テキストの指定	55
6	テキストのブロック・コピー／ムーブ	55
7	文字列のサーチとリプレイス	56
8	ラベルの検索	58
9	その他の機能	58
	■アン・ドゥ機能	59
	■タブ・サイズの変更	59
	■1行につなげる方法	59
	■ほかのファイルをテキスト内に追加する方法	60
	■画面表示色のカスタマイズ	60
	■プロシージャのウィンドウを開く	60
2-6	プログラムの実行	61
1	インタプリタによるプログラムの実行	61
2	実行可能ファイル(.EXE)の作成	62
	■コンパイラの起動	62
	■実行可能ファイルの作成形式	63
3	クイック・ライブラリの作成	64
	■クイック・ライブラリの作成	65
	■クイック・ライブラリの利用法	66
4	コマンド・ライン引数の取得	67

2-7 デバッグ	69
1 トレース機能	69
2 プログラムの低速実行	69
3 ウォッチ機能	70
4 ブレーク・ポイント	71
5 ウォッチ・ポイント	71
6 ダイレクトモード	72
7 実行の制御	73
8 ヒストリ	73
9 プロシージャ・コールの追跡	74
2-8 ヘルプ	75
1 オンラインヘルプ	75
2 全般的なヘルプ	76
2-9 QB モードから抜ける	77
1 QB を終了して MS-DOS に戻る	77
2 QB から MS-DOS に一時的に戻る (OS シェル)	77

第3章 コマンド・バージョン・ユーティリティ

3-1 ベーシック・コンパイラ(BC)	80
1 BC の起動法	80
■方法1 単独で起動する	80
■方法2 ファイル名をコマンド・ラインに与えて起動する	81
2 BC のコンパイル・オプション	82
3-2 リンカ(LINK)	85
1 LINK の起動法	85
■方法1 単独で起動する	86
■方法2 ファイル名をコマンド・ラインに与えて起動する	87
■方法3 応答ファイル名をコマンド・ラインに与えて起動する	87
2 LINK のリンカ・オプション	88
3-3 ライブラリ・マネージャ(LIB)	91
1 ライブラリ・マネージャとは	91
■ライブラリの新規作成	92
■ライブラリへの追加	92
■ライブラリからの削除	93
■ライブラリのモジュール置換	93
2 LIB の起動法	94
■方法1 単独で起動する	95

■方法2	ファイル名をコマンド・ラインに与えて起動する	95
■方法3	応答ファイル名をコマンド・ラインに与えて起動する	96

第4章 QuickBASICの言語仕様

4-1	プログラムの基本構成要素	98
1	QuickBASICの文字セット	98
2	プログラム行の構成	99
3	予約語	100
4-2	データ型	102
1	データ型の種類	102
2	定数	103
	■リテラル定数	103
	■記号定数	104
3	変数	104
	■名前のきまり	104
	■変数の型宣言	105
4	基本データ型	107
	■固定長文字列	108
	■文字列の格納イメージ	108
5	配列	109
	■配列の宣言	109
	■部分範囲指定	110
	■配列における留意事項	110
6	レコード型	111
	■レコード型の定義	112
	■レコード型変数の宣言	113
	■メンバの参照	113
7	通用範囲(スコープ)	114
	■ローカル変数とグローバル変数	114
	■モジュール間の変数の共有	115
	■プロシージャ間の変数の共有	115
	■DEF FN関数内のスコープ	116
8	記憶クラス	117
	■静的変数と自動変数	118
	■動的配列	119
9	混合演算と型変換	120

4-3	式と演算子	121
1	式	121
2	演算子の種類と優先順位	121
3	算術演算子	122
4	関係演算子	122
5	論理演算子	123
	■ビット演算	124
6	文字列演算	125
4-4	制御構造	127
1	構造化プログラミングと近代的流れ制御構造	127
2	判断	128
	■単純 IF 文(従来型 IF 文)	128
	■ブロック IF 文	128
	■複数条件判断(SELECT CASE 文)	129
3	くり返し	129
	■所定回反復(FOR 文)	129
	■前判定反復(WHILE~WEND 文, DO~LOOP 文)	130
	■後判定反復(DO~LOOP WHILE 文, DO~LOOP UNTIL 文)	131
4	とび越し	132
	■EXIT 文	132
	■GOTO 文	132
4-5	プロシージャ	133
1	従来型サブルーチンとプロシージャ	133
2	サブルーチン・プロシージャ	135
3	関数プロシージャ	136
	■サブルーチン・プロシージャと関数プロシージャの違い	137
4	プロシージャの留意事項	137
5	引数渡し	138
	■参照による呼び出し(Call by reference)	139
	■値による呼び出し(Call by value)	140
	■配列データを渡す	140
	■レコード・データを渡す	141
6	DEF FN 関数	142
7	再帰	142
4-6	メタコマンド	144
1	メタコマンドとは	144
2	\$INCLUDE	144
3	\$DYNAMIC,\$STATIC	144

第5章 ステートメントと関数の概要

5-1	実行制御	146
5-2	サブモジュール	147
5-3	変数管理	148
5-4	ファイル処理	149
	■ファイルの種類	150
	■ファイルのオープンとファイル番号	151
	■シーケンシャル・ファイルのリード／ライト	152
	■ランダム・ファイルのリード／ライト	152
	■デバイス・ファイル	154
5-5	コンソール入出力	155
	■画面構成	155
5-6	ディスク・ファイル操作	156
	■階層ディレクトリ	156
5-7	デバイス操作	157
5-8	文字列処理	158
	■漢字の扱い	159
5-9	数値処理	160
5-10	データ型の変換	161
	■マイクロソフト・バイナリ形式と IEEE 形式	162
5-11	エラー処理とトラッピング	163
	■エラー・トラッピング	163
	■イベント・トラッピング	164
	■複数モジュールでのトラッピング	165
	■コンパイル・オプション	166
5-12	時間	167
5-13	メモリ操作	168
	■セグメント値とオフセット値	168
5-14	DOS 環境	169
	■ソフトウェア割り込みとシステムコール	169
5-15	グラフィック	171
	■グラフィック座標	171
	■画面モード	172
	■表示色	173
	■アクティブ・ページとディスプレイ・ページ	174
5-16	その他	175

第6章 ステートメント・関数リファレンス

第7章 Ver.4.2から Ver.4.5への変更点

7-1	Ver.4.5における主な変更点	312
7-2	Ver.4.5で提供されるソフトウェア	313
7-3	セットアップ	317
	■ハードディスク	317
	■1MB フロッピー (SETUP を用いた場合)	319
	■1MB フロッピー (独自にコピーした場合)	320
7-4	メニュー画面	321
	■D/デバッグ	321
	■O/オプション	322
	■H/ヘルプ	324
7-5	エミュレータ版インタプリタ QBE.EXE	326
7-6	言語仕様上の変更点	327
	1 追加されたステートメント	327
	2 変更されたステートメント/関数	331
	3 環境変数 QBGRPNEC の導入	333
	4 [STOP]/[HELP]キーによるプログラム割り込みのサポート	333
	5 ソーステキスト, ライブラリの互換性	333
7-7	QB チュートリアル	334
7-8	HELPMAKE	335
	1 ヘルプファイルの概要	335
	2 ヘルプファイルの変更例	338
	3 HELPMAKE の仕様概要	339
7-9	ユーザライブラリ	341
	1 汎用ライブラリ	342
	2 グラフィック・ライブラリ	345
	3 マウスライブラリ	346
索引	351

第1章

Quick BASIC の概要

1-1

Quick BASIC の特徴

コンピュータ言語の中から BASIC をみた場合、言語仕樣的には決して優れたものであるとはいえません。たとえば、ブロック構造がない、近代的流れ制御構造が完備されていない、データ型が豊富でない、モジュール化ができないなどの点です。構造化プログラミングという概念からみると、ここに BASIC の限界があるといえます。

こうした状況において、Microsoft 社から Quick BASIC という処理系が発表されました。Quick BASIC は、

1. 統合開発環境
2. プログラムの構造化

という 2 つの強力な武器を備えた新世代言語です。

Quick BASIC は、国内では、PC-9800&AX シリーズ用の Ver.4.2 とバージョンアップされた Ver.4.5 が提供されています。

Quick BASIC Ver.4.2 のおもな特徴を箇条書きにまとめると、以下のようになります。

- ・高速で使い勝手のよい統合開発環境により、エディット、コンパイル、デバッグにおいて最適なプログラム開発環境が得られます。
- ・マウスを用いた快適なメニュー操作が可能です。
- ・統合開発環境からコンパイラ/リンカを起動して、実行可能ファイル(.EXE)を作成できます。
- ・スタンドオンライブラリ(.LIB)とクイックライブラリ(.QLB)という 2 種類のライブラリを利用することで有効な資源運用が図れます。
- ・コマンドバージョンのコンパイラ、リンカ、ライブラリ・マネージャをサポートします。
- ・MASM(マイクロソフト・マクロアセンブラ)、MS-C(Microsoft C Compiler)などのほかの言語とのインターフェースが簡単に行えます。
- ・ブロック概念の導入により、IF~THEN~ELSE 文などをスマートに書くことができます。
- ・DO~LOOP、SELECT CASE 文などの近代的流れ制御構造を補強し、構造的で見やすいプログラムが書けるようになりました。
- ・SUB、FUNCTION 文で定義したサブ・ルーチンと関数は、モジュールとして扱われます。モジュール間のデータ授受は引数渡しという汎用的な方法で行い、モジュール内の変数はローカル変数として扱われるので、独立性の高いモジュールを作ることができます。

- ・レコード型，記号定数，配列の部分範囲指定などによりデータ構造を明確に表現することができます。
- ・再帰呼び出し(リカーシブ・コール)を行うことができます。
- ・グラフィック処理のためのステートメントが提供されています。

なお，Ver.4.5への変更点については，第7章で説明します。詳しくはそちらをごらんください。

1-2

システムの構築

1 提供されるソフトウェア

Quick BASIC Ver.4.2は、プログラム・ディスクとライブラリ&サンプルプログラムの2枚のフロッピーディスクで提供されています。また、バージョンアップしたQuick BASIC Ver.4.5は、セットアップ・ディスクとプログラム・ディスク、ライブラリ・ディスクそしてアドバイザー・ディスクの4枚のフロッピーディスクが提供されています。

●プログラム・ディスク

ドライブ A: のディスク のボリュームラベルは QB42_N01
ディレクトリは A:¥

QB_ENV	<DIR>	88-12-15	4:20	←統合環境ディレクトリ	
QB_CMP	<DIR>	88-12-15	4:20	←コンパイラ・ディレクトリ	
LIB_E	<DIR>	88-12-15	4:20	←ライブラリ・ディレクトリ	
SETUP	DOC	2763	88-12-15	4:20	←セットアップの説明
SETUP	EXE	30198	88-12-15	4:20	←セットアップ・ユーティリティ
PACKING	LST	3242	88-12-15	4:20	←パッキング・リスト
README	DOC	8457	88-12-15	4:20	←最新情報ファイル
SAMPLE	DOC	4043	88-12-15	4:20	←サンプルプログラムの説明
8 個のファイルがあります。			4:20		
175104 バイトが使用可能です。					

▼統合開発に関するシステム

ドライブ A: のディスク のボリュームラベルは QB42_N01
ディレクトリは A:¥QB_ENV

.	<DIR>	88-12-15	4:20		
..	<DIR>	88-12-15	4:20		
MOUSE	COM	8431	88-12-15	4:20	←マウス・ドライバ
MOUSE	SYS	8127	88-12-15	4:20	←マウス・ドライバ
QB	BI	1783	88-12-15	4:20	← Quick BASIC ヘッドファイル
QB	EXE	284170	88-12-15	4:20	← Quick BASIC 本体
QB	HLP	60519	88-12-15	4:20	←ヘルプ・ファイル
QB	INI	48	88-12-15	4:20	←イニシャル・ファイル
QB	PIF	369	88-12-15	4:20	← MS-WINDOWS2.0用 Pictur File
9 個のファイルがあります。					
175104 バイトが使用可能です。					

QB_CMP <DIR>

▼コンパイラに関するシステム

ドライブ A: のディスク のボリュームラベルは QB42_N01
ディレクトリは A:\QB_CMP

.	<DIR>	88-12-15	4:20	
..	<DIR>	88-12-15	4:20	
BC	EXE	110743	88-12-15	4:20 ← BASIC コンパイラ
BRUN42A	EXE	88539	88-12-15	4:20 ←ランタイムモジュール
BRUN42E	EXE	89328	88-12-15	4:20 ←ランタイムモジュール
LIB	EXE	35211	88-12-15	4:20 ←ライブラリ・マネジャー(ライブラリアン)
LINK	EXE	66539	88-12-15	4:20 ←リンカ

7 個のファイルがあります。
175104 バイトが使用可能です。

LIB_E <DIR>

▼ライブラリ

ドライブ A: のディスク のボリュームラベルは QB42_N01
ディレクトリは A:\LIB_E

.	<DIR>	88-12-15	4:20	
..	<DIR>	88-12-15	4:20	
BCOM42E	LIB	234477	88-12-15	4:20 ←スタンドアロン・ライブラリ
BRUN42E	LIB	25241	88-12-15	4:20 ←分離型ライブラリ

4 個のファイルがあります。
175104 バイトが使用可能です。

●ライブラリ&サンプルプログラム・ディスク

ドライブ A: のディスク のボリュームラベルは QB42_N02
ディレクトリは A:\

LIB_A	<DIR>	88-12-15	4:20	
SOURCE	<DIR>	88-12-15	4:20	

2 個のファイルがあります。
820224 バイトが使用可能です。

▼ライブラリ

ドライブ A: のディスク のボリュームラベルは QB42_N02
ディレクトリは A:\LIB_A

.	<DIR>	88-12-15	4:20	
..	<DIR>	88-12-15	4:20	
BCOM42A	LIB	238895	88-12-15	4:20 ←スタンドアロン・ライブラリ
BQLB42	LIB	25301	88-12-15	4:20 ←QLB 構築ライブラリ
BRUN42A	LIB	25241	88-12-15	4:20 ←分離型ライブラリ
QB	LIB	2075	88-12-15	4:20 ←ライブラリ
QB	QLB	5784	88-12-15	4:20 ←Quick ライブラリ

7 個のファイルがあります。
820224 バイトが使用可能です。

ドライブ A: のディスクのボリュームラベルは QB42_N02
ディレクトリは A:\SOURCE

```
.          <DIR>      88-12-15    4:20
..         <DIR>      88-12-15    4:20
BAR        BAS        6331    88-12-15    4:20
CAL        BAS        5595    88-12-15    4:20
CCALL      BAS         955    88-12-15    4:20
CCALL      C          100    88-12-15    4:20
CCALL      QLB        5351    88-12-15    4:20
COLOR      BAS        1019    88-12-15    4:20
CRLF       BAS        4441    88-12-15    4:20
CUBE       BAS        1413    88-12-15    4:20
DISPLAY    BAS        1829    88-12-15    4:20
DRAW       BAS        2376    88-12-15    4:20
EDIT       BAS        8675    88-12-15    4:20
EDIT       BI         1308    88-12-15    4:20
EDPAT      BAS        6667    88-12-15    4:20
GET        BAS         842    88-12-15    4:20
HANOI      BAS        1289    88-12-15    4:20
INDEX      BAS       10366    88-12-15    4:20
MANDEL     BAS        5560    88-12-15    4:20
PALETTE    BAS        1797    88-12-15    4:20
PSETBALL   BAS        3000    88-12-15    4:20
QBL        BAS        6902    88-12-15    4:20
QLBDUMP    BAS        2783    88-12-15    4:20
SEARCH     BAS        3028    88-12-15    4:20
SIERP      BAS        2119    88-12-15    4:20
SINWAVE    BAS        1130    88-12-15    4:20
STRTONUM   BAS         999    88-12-15    4:20
STYLE      BAS         387    88-12-15    4:20
TERMINAL   BAS        1343    88-12-15    4:20
WHEREIS    BAS        5453    88-12-15    4:20
ABSOLUTE   ASM        4520    88-12-15    4:20
INTRPT     ASM       15099    88-12-15    4:20
```

32 個のファイルがあります。
820224 バイトが使用可能です。

Quick BASIC Ver.4.2が提供するソフトウェアのおもなものを、機能別に分類すると以下ようになります。なお、Ver.4.5については第7章をごらんください。

表1-1 Quick BASIC 統合開発環境 QB

ファイル名	機 能
QB. EXE	Quick BASIC 本体、エディタ、インタプリタ、デバッガ内蔵 ヘルプ・ファイル 画面の初期化情報ファイル QB.QLB(Quick ライブラリ)用インクルードファイル
QB. HLP	
QB. INI	
QB. BI	

表1-2 BASIC コンパイラ BC

ファイル名	機 能
BC. EXE LINK. EXE LIB. EXE BRUN42A. EXE BRUN42E. EXE	BASIC コンパイラ リンカ ライブラリ・マネジャ (ライブラリアン) ランタイム・モジュール(代替数値演算版) ランタイム・モジュール(80?87エミュレート版)

表1-3 ライブラリ

ファイル名	機 能
BRUN42A. LIB BCOM42A. LIB BRUN42E. LIB BCOM42E. LIB	ランタイム・ライブラリ (代替数値演算版) スタンドアロン・ライブラリ(//) ランタイム・ライブラリ (80?87エミュレート版) スタンドアロン・ライブラリ(//)
QB. QLB QB. LIB	デフォルトの Quick ライブラリ デフォルトのライブラリ

表1-4 その他

ファイル名	機 能
SETUP. EXE MOUSE. COM MOUSE. SYS	Quick BASIC のセットアップ・ユーティリティ マウス・デバイス・ドライバ(常駐/非常駐型) マウス・デバイス・ドライバ(組み込み型)

2 環境の設定

■メモリの設定

Quick BASIC を動作させるためには640KB のメモリが必要です。メモリが足らなければ、PC-9801の機種に応じて適当にメモリを増設してください。

メモリ増設をした際には、MS-DOS の SWITCH コマンドを用いてメモリ・サイズの指定を行ってください。SWITCH コマンドを行ったあと、一度リセットをしてください。メモリの設定は一度しておけば、それが記憶されています。

なお、PC-9801本体のディップスイッチ SW2の No.5が ON になっていなければなりません。


```
A>switch

SWITCH  Version 2.30

RS232C-0:1200  BITS-7  PARITY-NONE  STOP-1  NONE
PRINTER :CEN24
MEMORY  :640K
NDP1    :NO
COLOR   :WHITE
BOOT    :STD

-memory[640]
```

■パスの設定

Quick BASIC は、次の 4 つの MS-DOS 環境変数を参照して処理を行います。

表1-5 MS-DOS の環境変数

環境変数	機 能
COMSPEC	再ロードするコマンド・インタプリタのパス名を設定しておく Quick BASIC が OS シェル(OS に一時的に戻る)するときに、この環境変数に設定されているコマンド・インタプリタをロードする
PATH	コマンドはカレントディレクトリから探されるが、これ以外から探すときの道筋を PATH 環境変数に設定する
LIB	ライブラリの検索パスを設定する
TMP	LINK が作業に使うテンポラリファイルを置く位置を設定する RAM ディスクがあれば、そこに置くようにすると高速処理が行える

■日本語 FEP の組み込み

Quick BASIC で使用できる日本語 FEP(フロント・エンド・プロセッサ)は次の 2 種類です。

- ・ ATOK6 または ATOK7
- ・ MS-KANJI API 仕様の日本語入力システム

ここでは ATOK6 を組み込む方法を説明します。ATOK6 を使用するには、次のファイルが必要です。

- ATOK6A.SYS
 - ATOK6B.SYS
 - ATOK.DIC
- └─ かな漢字変換デバイス・ドライバ
- 辞書ファイル

Quick BASIC のインタプリタ/コンパイラシステムを1MB フロッピーにセットアップした場合、ディスク容量がわずかしかなかったため、日本語システムはドライブ B にのせなければなりません。この場合の CONFIG.SYS での設定は、次のように記述します。

● CONFIG. SYS 内の記述

DEVICE = B : ATOK6A. SYS /D = B
DEVICE = B : ATOK6B. SYS

■マウス・ドライバの組み込み

Quick BASIC では次の 2 つのマウス・ドライバを提供しています。

表1-6 マウス・ドライバの種類

マウス・ドライバ	機 能
MOUSE. SYS	従来のデバイス・ドライバで、CONFIG.SYS の中で DEVICE = MOUSE. SYS と指定することでシステム起動時に組み込まれる。起動後はマウス・ドライバをシステムから切り離すことはできない
MOUSE. COM	コマンドラインからコマンドを投入することにより、マウス・ドライバを常駐/非常駐させることができる。コマンド書式は次のとおりである MOUSE バス・マウスを利用する MOUSE /C シリアル・マウスを利用する MOUSE OFF MOUSE. COM をメモリから解放する したがってこの MOUSE.COM は AUTOEXEC. BAT の中でシステム起動時に自動実行させて組み込む

SETUP ユーティリティではどちらのマウス・ドライバを使用するのか問い合わせてきますので、MOUSE.SYS または MOUSE.COM を選択します。

MOUSE.COM のほうが、常駐/非常駐の制御をユーザが行える点で便利だと思います。

■プリンタ・ドライバの組み込み

従来の MS-DOS では、プリンタ・デバイス・ドライバは MS-DOS システムの中にあらかじめ組み込まれていました。ところが、MS-DOS Ver.3.3と MS-DOS Ver. 3.1の一部(製品番号 PS98-011-×××)では、PRINT.SYS という専用のプリンタ・デバイス・ドライバとして MS-DOS から分離しています。したがってこの場合は、ユーザが CONFIG.SYS の中で PRINT.SYS を登録しなければなりません。

● CONFIG. SYS 内の記述(MS-DOS Ver.3.1の一部と Ver.3.3)

DEVICE = PRINT. SYS

■ライブラリの選択

Quick BASIC のコンパイラ (BC.EXE) / リンカ (LINK.EXE) で利用するライブラリは、

- ・生成する実行可能ファイルのタイプ
- ・数値演算コ・プロセッサ(8087/80287)の使用の有無

の違いにより、それぞれ次のように異なるライブラリを使います。

表1-7 コンパイラ用ライブラリの種類

コ・プロセッサ EXE のタイプ	代替数値演算	8087/80287エミュレート
スタンドアロン型	BCOM42A. LIB	BCOM42E. LIB
ランタイム分離型	BRUN42A. EXE BRUN42A. LIB	BRUN42E. EXE BRUN42E. LIB

スタンドアロン型の EXE ファイルは単独で動作しますが、ランタイム分離型の EXE ファイルは実行時にランタイム・モジュール (BRUN42A.EXE または BRUN42E.EXE) をロードし、このモジュールの協力下において動作します。

代替数値演算ライブラリは、数値演算コ・プロセッサを使用せずに数値演算を行います。8087/80287エミュレート・ライブラリは、数値演算コ・プロセッサを使用して数値演算を行います。

統合開発環境 QB では一般にライブラリを必要としませんが、QB.QLB, QB.LIB などのライブラリを使用することもできます。

3 システムの構築例

■SETUP ユーティリティ

Quick BASIC は SETUP ユーティリティを用いて簡単にフロッピーディスクまたはハードディスクにシステムを構築できます。ただし、MS-DOS システムをあらかじめコピーしておきます。

A>SETUP

Quick BASIC セットアップ ユーティリティ Ver 1.00

---準備---

(1/3)

◎フロッピーディスクへセットアップする場合、フォーマット済みのディスクを用意してください。

〔フロッピーベースでQuick BASICを利用する方へ〕

あらかじめフォーマット済みのディスクが必要です。

メディアのタイプによって必要なディスクの枚数が異なります。

ワークディスクには FORMAT /S スイッチでMS-DOSシステムを転送しておいて下さい。システムが転送されていない場合はセットアップはおこなわれません。

・HDタイプ(3.5 2HD, 5 2HD)の場合

1枚：ワークディスク

・DDタイプ(3.5 2DD, 5 2HD)の場合

2枚：ワークディスク、コンパイラディスク

(FORMAT /9 で 720Kフォーマットにしておく方が良いでしょう。)

◎フォーマットが済みましたら添付のシール(「ワークディスク」, 「コンパイラディスク」)をディスクに貼っておいて下さい。他のシールは使いません。御自由にお使い下さい。

フォーマット済みのディスクが用意していなければ、CTRL+C でセットアップを中止してディスクのフォーマットを行って下さい。

【何かキーを押して下さい】 ←何かキーを押すと次の画面に進む。

Quick BASIC セットアップ ユーティリティ Ver 1.00

---サンプルプログラム---

(2/3)

セットアップでは、サンプルプログラムを転送しません。必要に応じて転送をおこなって下さい。

---ライブラリについて---

Quick BASICのコンパイラ(BC.EXE)が通常利用するライブラリは、以下のものです。

BRUN42A.EXE ランタイムモジュール

BRUN42A.LIB ランタイムライブラリ

BCOM42A.LIB スタンドアロンライブラリ

これらを代替数値演算ライブラリ(ALT-MATH)と呼びます。これらは、数値演算コプロセッサの有無に関わらず、一定の速度で高精度な計算を実現します。

数値演算コプロセッサを装着している方のため BRUN42E.EXE, BRUN42E.LIB, BCOM42E.LIB (80?87エミュレートライブラリ)を用意しています。これによってより高度な演算を実現できます。

フロッピーベースでご利用になる場合は、セットアップではエミュレートライブラリは転送できません。必要な場合は、マスターディスクから転送して下さい。

【何かキーを押して下さい】 ←何かキーを押すと次の画面へ進む

---フロントプロセッサの組み込み---

(3/3)

QuickBASICに日本語入力システムを組み込む場合は、お使いのフロントプロセッサのマニュアルを参照して各自おこなってください。

なお、ご利用可能なフロントプロセッサは、ATOK6、MS-KANJI API のものです。添付の「日本語入力システムのご使用について」を参考にしてください。

---CONFIG.SYS、AUTOEXEC.BAT---

このセットアッププログラムは、フロッピーへセットアップしたとき「ワークディスク」に CONFIG.SYS、AUTOEXEC.BAT を作成します。ハードディスクの場合は、指定されたディレクトリに、NEW-CONF.SYS、NEW-PATH.BAT を作成しますので、これらのファイルを参考にして CONFIG.SYS、AUTOEXEC.BAT を修正してください。環境変数については、README.DOC をご参照下さい。

なお、README.DOC には補足の情報や、訂正資料が記載されています。必ずお読み頂くようお願いします。

【何かキーを押して下さい】 ←何かキーを押すと次の画面に進む。

- ハードディスクにセットアップしますか [Y/N] ? N
- QuickBASICをどのドライブにインストールしますか ? B
- マウスを使用しますか [Y/N] ? Y
- マウスの種類を選択して下さい [1:ハス/2:シリアル] ? 1
- マウスドライバを選択して下さい [1:MOUSE.SYS/2:MOUSE.COM] ? 2

各項目に答える

上記の設定でよろしいですか [Y/N] ? Y ←Yと答えるとSETUP作業が開始される

マウスをサポートするデバイスドライバの種類を選択して下さい。
メモリ内に常駐するタイプが1 (MOUSE.SYS) で、必要に応じて組込みが可能なタイプが2 (MOUSE.COM) です。
MOUSE.COMは、MOUSE OFF でメモリ解放ができる為、COM形式を使うことをお勧めします。

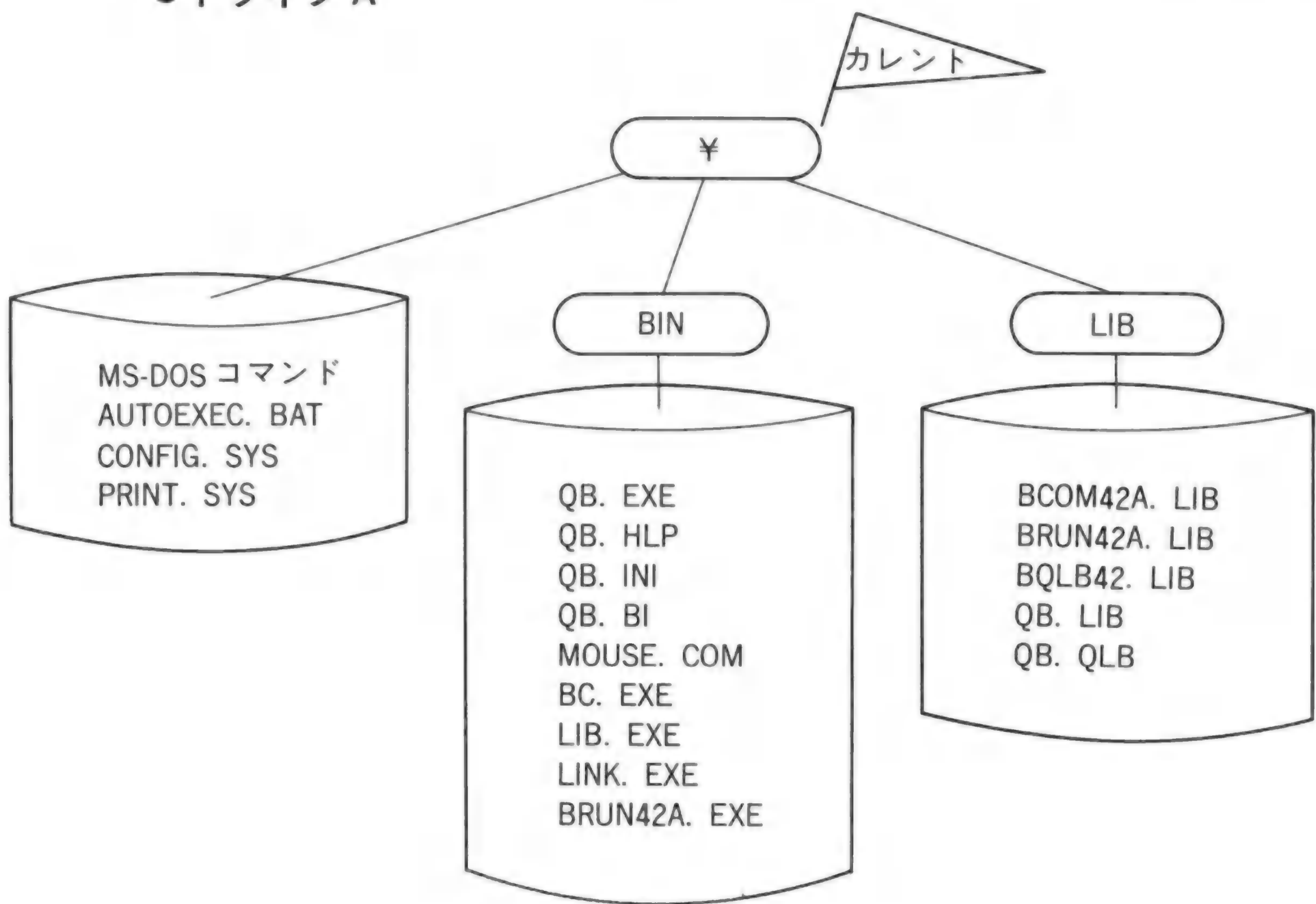
■1MB フロッピーでの構築例

日本語辞書をドライブ B に置きます。

マウス・ドライバは MOUSE.COM を選択します。

ソースファイルはドライブ B に作っていきます。

● ドライブ A

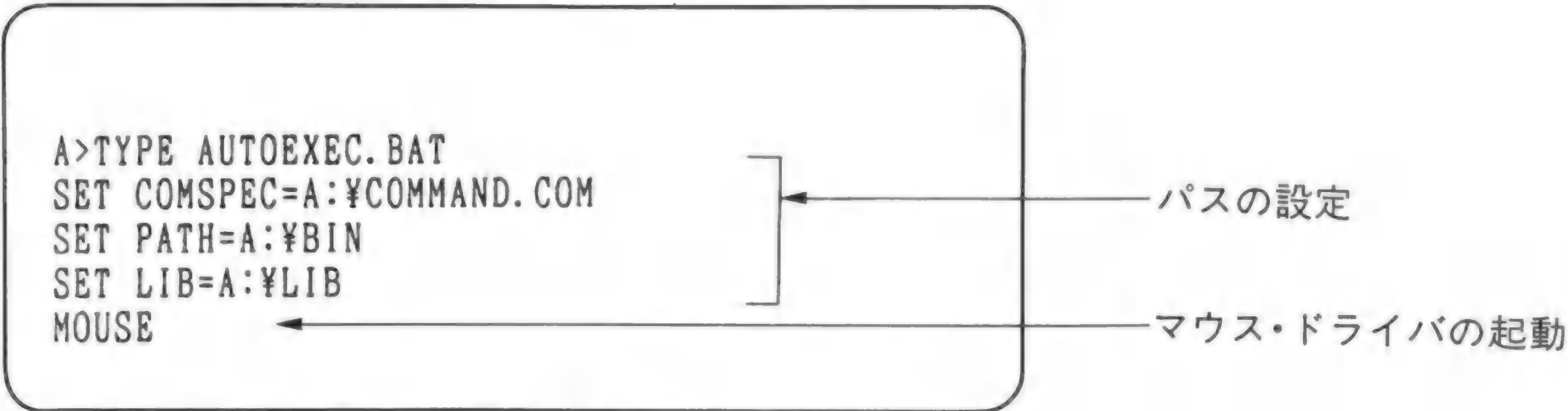


● ドライブ B

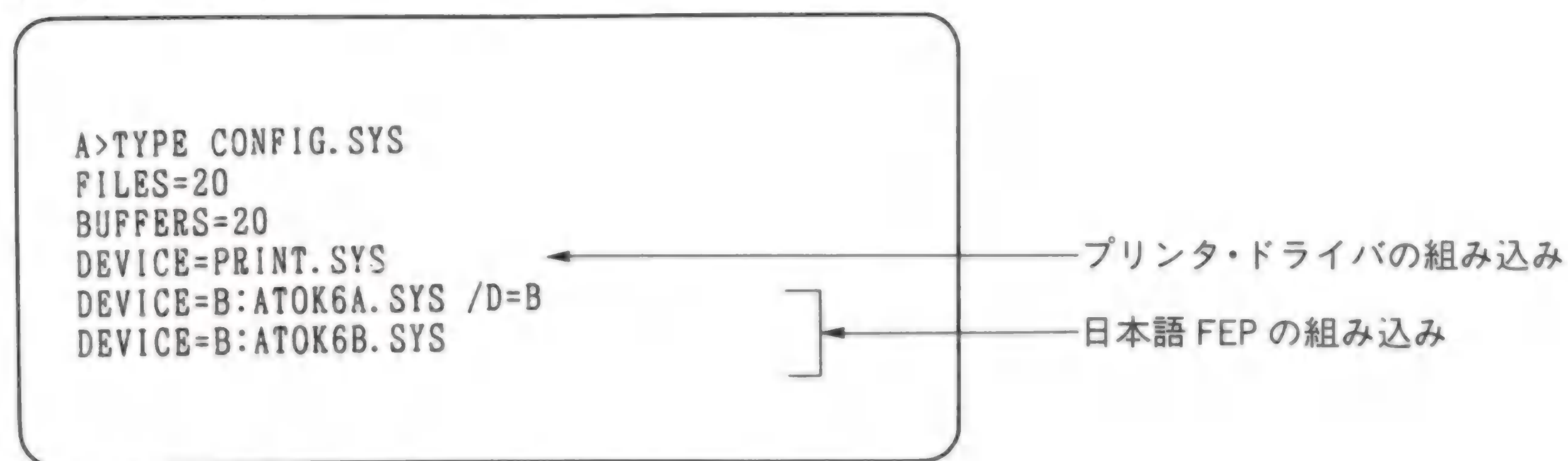


このようなシステム環境では、AUTOEXEC.BAT と CONFIG.SYS を次のように作ります。

● AUTOEXEC. BAT



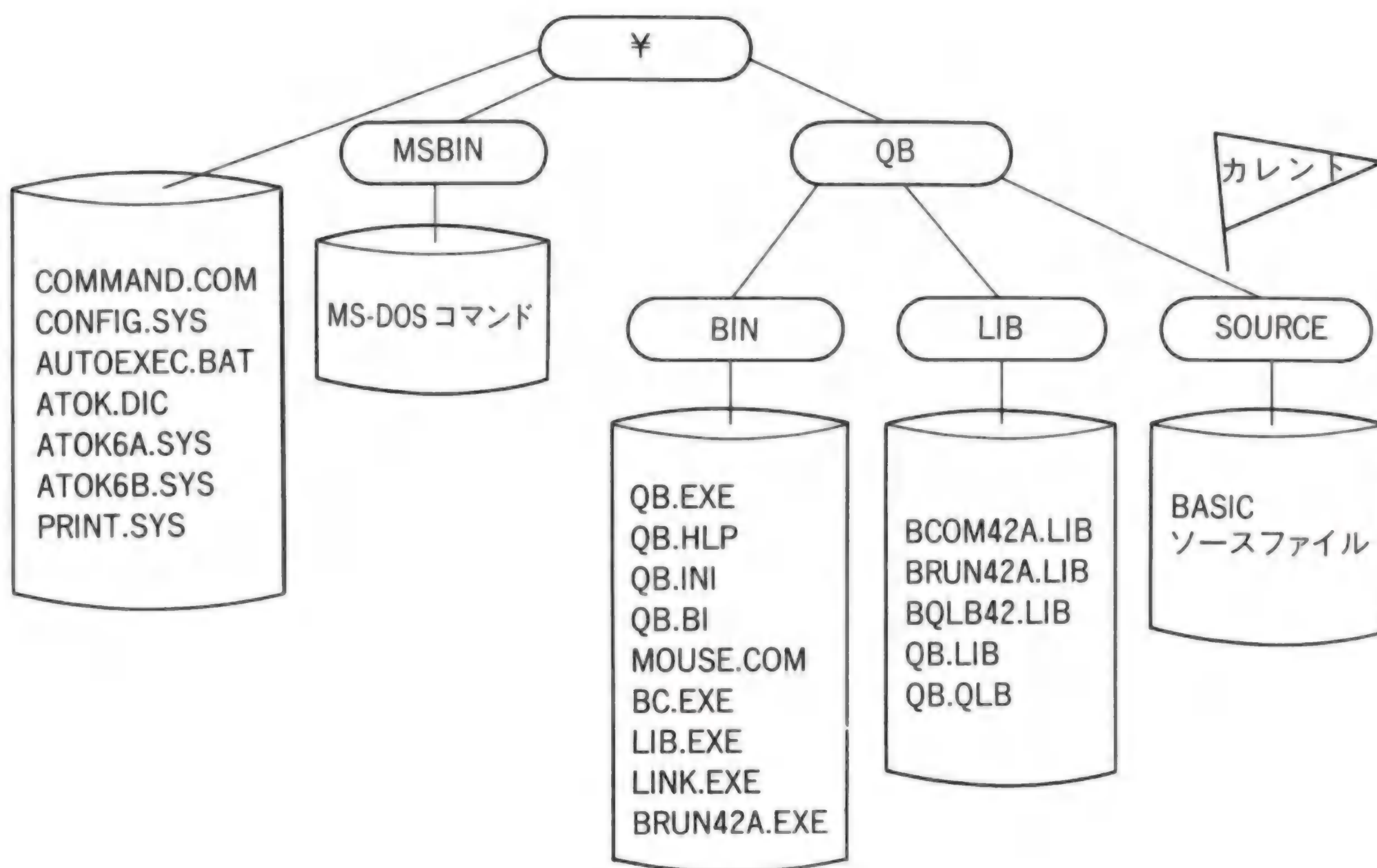
● CONFIG. SYS



注) BASIC コンパイラとライブラリを除いて QB システムだけにすれば, ATOK 関係をドライブ A にのせることができます。

■ ハードディスクでの構築例

ハードディスクのような大容量システムでは, Quick BASIC 以外の言語もディスク上にいっしょにのせることが考えられるので, Quick BASIC のシステムは, QB というサブディレクトリを設け, その下に置くことにします。また, ユーザファイルもハードディスク上に置いたほうが有利なので, これを SOURCE というサブディレクトリに置き, ここをカレントディレクトリにします。このような点を考慮したハードディスクでのシステム構築例を, 以下に示します。



このようなシステム環境では、AUTOEXEC.BAT と CONFIG.SYS を次のように作ります。

● AUTOEXEC. BAT

```
SET COMSPEC=%COMMAND.COM
SET PATH=%QB%BIN;%MSBIN;%
SET LIB=%QB%LIB
MOUSE
CD %QB%SOURCE
```

● CONFIG. SYS

```
FILES=20
BUFFERS=20
DEVICE=ATOK6A.SYS
DEVICE=ATOK6B.SYS
DEVICE=PRINT.SYS
```


1-3

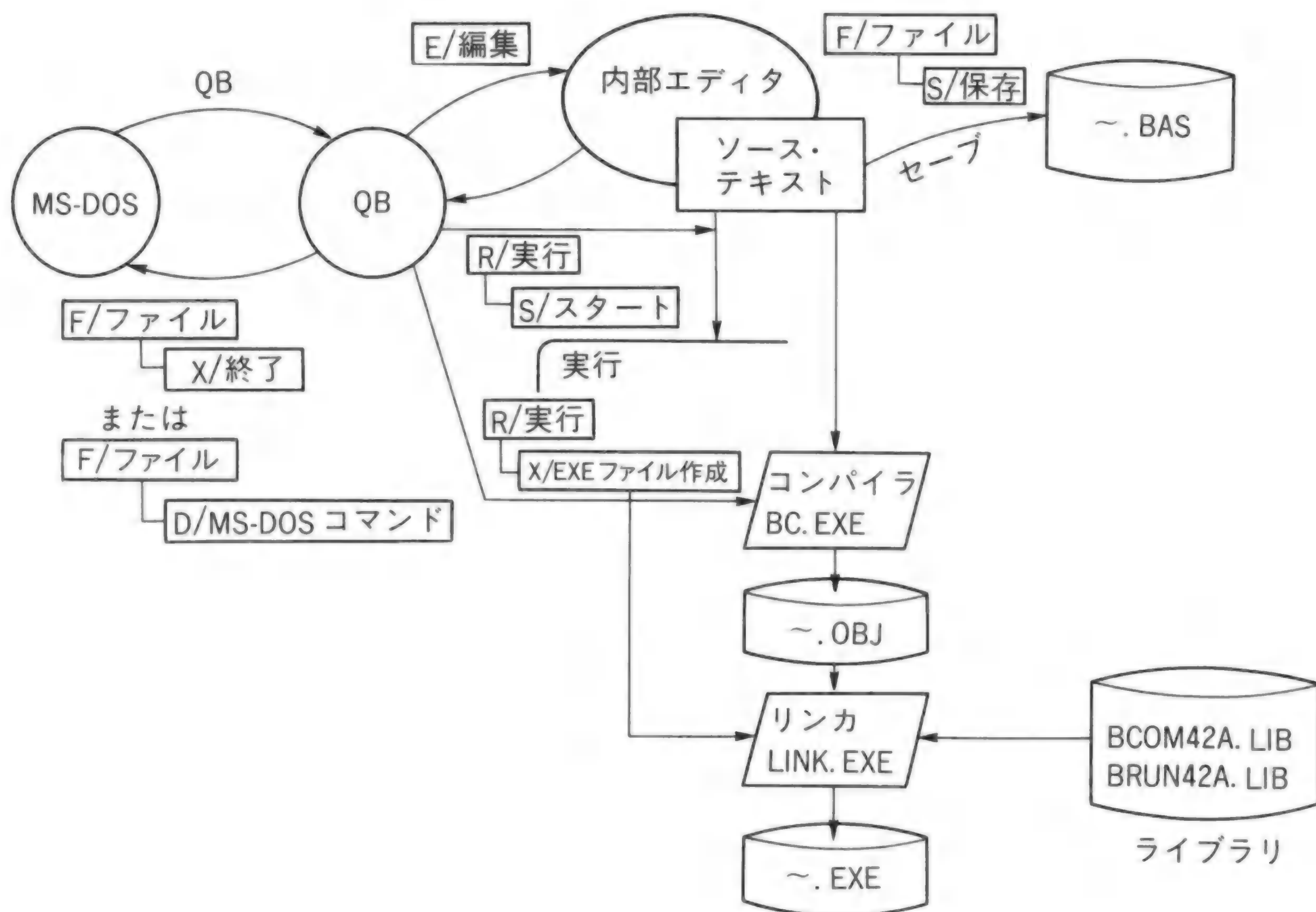
Quick BASIC の運用形態

提供される Quick BASIC のシステム・ディスクの中には、統合開発環境 QB と BASIC コンパイラ BC の2つのシステムが入っています。

統合開発環境は Quick BASIC の基本システムで、QB.EXE を本体とし、この中にエディタ、インタプリタ、デバッガが内蔵されています。一度 QB を起動すれば、プル・ダウン・メニュー方式で各処理を高速に行うことができます。

QB は基本的にはインタプリタです。しかしプル・ダウン・メニューから BASIC コンパイラ (BC.EXE) とリンカ (LINK.EXE) を自動的に呼び出し、実行可能ファイル (.EXE) を生成することもできます。

BC.EXE, LINK.EXE は従来のコンパイラ処理系と同様に、コマンドラインからコンパイル/リンク処理を行うこともできます。



第2章

統合開発 環境(Q B)

2-1

QB の起動とメニュー画面

QB の起動は、

A>QB ☐

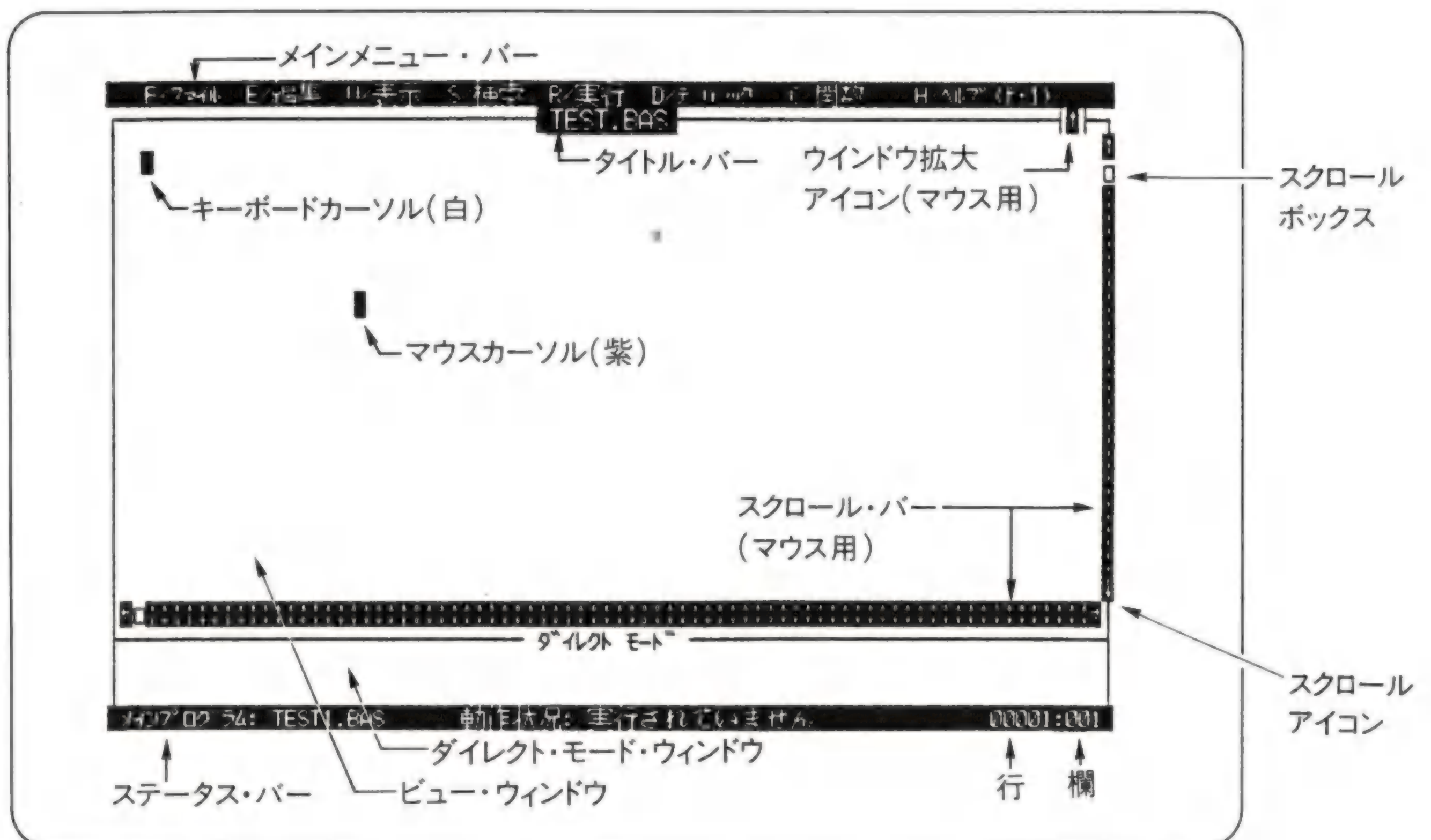
と単独で起動し、ファイルセーブ時にファイル名を指定するか、

A>QB ☐ B : TEST ☐

↑ソースファイル名。ファイルタイプを省略すると.BAS とみなされる。

とソースファイル名を指定して起動します。

これにより、次のようなメニュー画面が表示されます。



■ QB の起動オプション

QB の起動時に、次のようなオプションを指定することができます。オプション文字の大/小は区別されません。

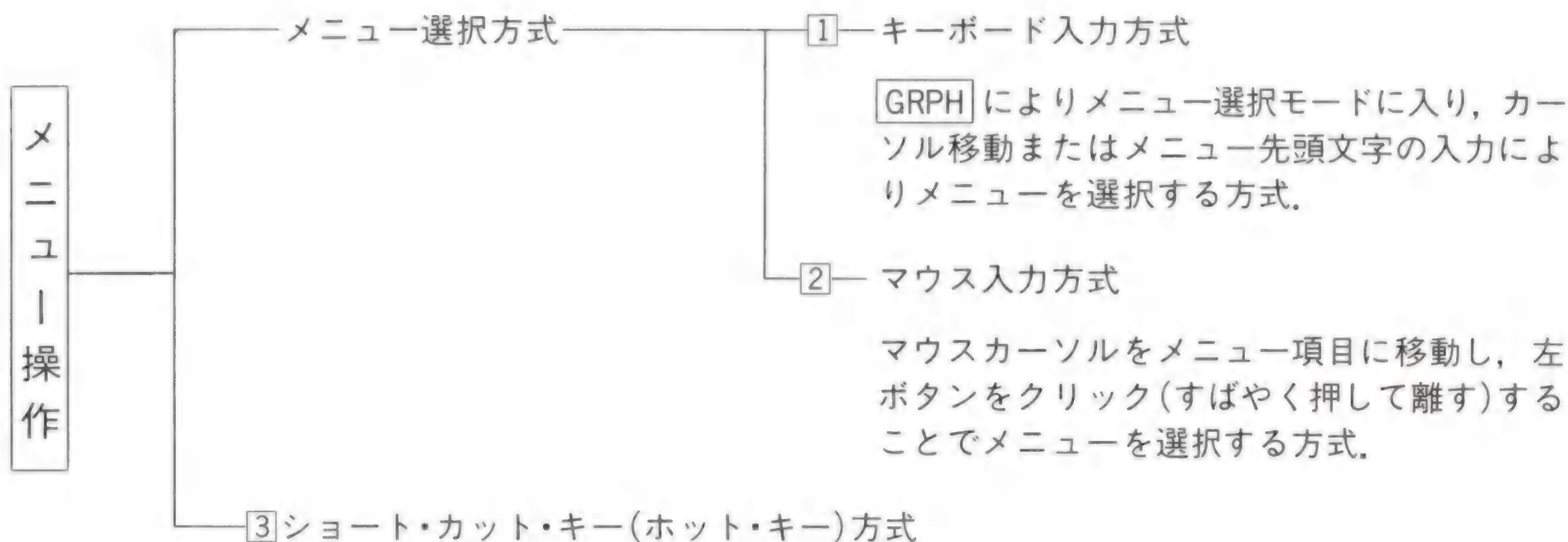
オプション	機 能
/AH	動的配列に対するメモリの解放(第 3 章3-1の 2 参照)
/B	モノクロ画面表示にする
/C:n	通信ポート・バッファのサイズの指定(第 3 章3-1の 2 参照)
/CMD string	string で示される文字列を COMMAND\$関数に引き渡す。このオプションはコマンドラインの最後に指定する
/L[qlib]	qlib で示されるクイックライブラリを QB システムに読み込む。qlib を省略すると QB.QLB が読み込まれる
/MBF	IEEE 形式の数値をマイクロソフトバイナリ形式の数値として扱う(第 3 章3-1の 2 参照)
/NOHI	2 階調表示のディスプレイを使うときに/B オプションと組み合わせて指定する
/RUN sourcefile	sourcefile で示されるソースファイルを QB 起動時に読み込み, 実行する

2-2 Quickメニューの使い方

1 メニュー操作の基本

■メニュー操作の方式

メニュー操作は、おおよそ次の3つの方法のいずれかで行うことができます。



メニューの一部のものはファンクションキー(CTRL, SHIFTと組み合わせる場合もある)に割り当てられているので、これを1タッチすることでメニューを選択する方式。

■メニュー操作で使うキー

メニュー操作で使うおもなキーは以下のものです。

表2-1 メニュー操作に必要なキー

メニュー操作キー	機 能
<div>GRPH</div>	メニュー選択モードに入る、メニュー項目の先頭文字の色が変わる メニュー選択モードから抜けるには再び <div>GRPH</div> を押す メインメニューの選択では <div>GRPH</div> を押し続ける必要はないが、サブメニュー内では押しながらメニュー選択を行う必要がある場合がある
<div>TAB</div>	サブメニュー内の項目を次に進める
<div>SHIFT</div> + <div>TAB</div>	サブメニュー内の項目を 1 つ前に戻す
<div>↵</div>	メニューバー(カーソル)の置かれているメニューを実行する
<div>←</div> <div>→</div>	メインメニュー・バーを移動する
<div>↑</div> <div>↓</div>	サブメニュー・バーを移動する
<div>ESC</div>	メニュー処理を取り消す

■マウス入力方式

●メニューの選択

マウス・カーソルをメニューに合わせ左ボタンをクリック(マウスボタンをすばやく押して離すこと)します。

メインメニューを取り消すには、メニュー枠の外にカーソルを移し、左ボタンをクリックします。

サブメニューを取り消すには、

取消

にカーソルを移し、左ボタンをクリックします。

● コマンドの選択

マウス操作	機能
ウィンドウのクリック	ウィンドウをアクティブにする
カーソルをタイトルバーに移し、タイトルバーを上下に移動	アクティブウィンドウの拡大/縮小
拡大アイコン(↑)をクリックするか、タイトルバーをダブルクリック	アクティブウィンドウを画面一杯に拡大する
スクロールボックス (□) にカーソルを移し、スクロールバー上で目的位置まで左ボタンを押しながら移動	テキストのスクロール
スクロールバー上の□で分離された上半分または下半分のクリック	1 ページ単位の上/下スクロール
スクロールアイコン(↑, ↓, ←, →)をクリック	1 行または 1 文字単位の上/下, 左/右のスクロール

■ サブメニューの意味

サブメニューの各項目の一般的な意味を示します。

メインメニューのメニュー・バー

F/ファイル E/編集 U/表示 S/検索

サブメニューのメニュー・バー

L/元に戻す GRPH+BS

T/カット SHIFT+DEL

C/コピー CTRL+INS

P/ペースト SHIFT+INS

E/削除 DEL

S/新規SUB...

F/新規FUNCTION...

> Y/構文チェック

対応するショート・カット・キー

先頭の文字の色が変わっていないものは、今の状態では実行できないことを示す

...があるサブメニューは、この下にさらにサブメニューがあることを示す

>があると、この項目が示す機能が ON 状態にあることを示す
この項目を選択するたびに ON/OFF のトグル動作となる

注) Ver. 4.5 では、>ではなく*の表示

2 Quick メニュー一覧

Quick BASIC のメインメニューとサブメニューの一覧を次に示します。

表2-2 メニュー一覧

メインメニュー	サブメニュー	機能	ショート・カット・キー	解説項
F ／ ファイル	N/新規	メモリ上のプログラムを消去する		2-4の2
	O/読込...	メモリ上に新しいプログラムをロードする		2-4の3
	M/追加...	メモリ上に別のプログラムを追加ロードする		//
	S/保存	アクティブウィンドウのプログラムをセーブする		2-4の1
	A/名前を変えて保存...	アクティブウィンドウのプログラムを別のファイル名, 別の保存モードでセーブする		//
	V/すべて保存	メモリ上のすべてのモジュールをセーブする		//
	C/サブファイル作成...	新しいモジュール/インクルードファイル/ドキュメントファイルを作成する		2-4の5 6, 7
	L/サブファイル常駐...	モジュール/インクルードファイル/ドキュメントファイルをメモリ上にロードする		//
	U/サブファイル解放...	メモリ上のモジュール/インクルードファイル/ドキュメントファイルを消去する		//
	P/印刷...	メモリ上のプログラムをプリンタに印刷する		2-4の4
	D/MS-DOS コマンド	一時的に MS-DOS に抜ける, OS シェル		2-9の2
	X/終了	QB モードを終了して MS-DOS に戻る		2-9の1
E ／ 編集	U/元に戻す	最後の編集動作を行う前の状態に戻す アンドゥ機能	GRPH + BS	2-5の9
	T/カット	指定されたテキストを削除し, クリップボードに格納する	SHIFT + DEL	2-5の6
	C/コピー	指定されたテキストをクリップボードに格納する	CTRL + INS	//
	P/ペースト	クリップボードの内容を, 現在のカーソル位置に挿入する	SHIFT + INS	//
	E/削除	指定されたテキストを削除する	DEL	2-5の5
	S/新規 SUB...	新しい SUB プロシージャのウィンドウを開く		2-5の9
	F/新規 FUNCTION...	新しい FUNCTION プロシージャのウィンドウを開く		//
	Y/構文チェック	自動構文チェック機能の ON/OFF を切り換える		2-5の3

メインメニュー	サブメニュー	機能	ショート・カット・キー	解説項
V ／ 表示	S/SUB 一覧...	メモリ上の SUB プロシージャの一覧を表示する	F・2	2-4の5
	E/次の SUB	アクティブウィンドウに, 次のサブプログラムを表示	SHIFT + F・2	2-4の5
	P/画面分割	画面分割と単一ウィンドウを切り換える		2-3の1
	N/次のステートメント	次に実行されるステートメントを表示する		2-7の7
	U/実行画面	出力画面に切り換える	F・4	2-3の4
	I/インクルードファイル編集	インクルードファイルを表示して編集モードに入る		2-4の7
	L/インクルードファイル表示	インクルードファイルを表示する		//
	O/オプション...	画面の表示色, タブ・サイズの設定をする		//
S ／ 検索	F/検索...	指定したテキストを探す		2-5の7
	S/指定文字列検索	アクティブウィンドウ中で反転表示されているテキストを検索する	CTRL + ¥	//
	R/次を検索	次に一致する検索文字列を探す	F・3	//
	C/置換...	指定したテキストを探し新しいテキストに置き換える		//
	L/ラベル...	ラベルの検索を行う		
R ／ 実行	S/スタート	プログラムを実行する	SHIFT + F・5	2-6の1
	R/リスタート	プログラム中の変数をクリアし, プログラム先頭を実行開始点に設定する		2-7の7
	N/続行	STOP 文などにより中断されていたプログラムを続行する	F・5	//
	C/COMMAND\$入力...	COMMAND\$関数が返す文字列を設定する		2-6の4
	X/EXE ファイル作成...	コンパイラ/リンカを起動し, 実行可能ファイル(.EXE)をディスクに作成する		2-6の2
	L/ライブラリ作成...	コンパイラ/リンカ/ライブラリを起動し, クィック・ライブラリをディスクに作成する		2-6の3
	M/メインモジュールの設定...	実行の開始点になるメインモジュールを他へ移す		
D ／ デバ ッグ	A/ウォッチの追加...	ウォッチ式を設定する		2-7の3
	W/ウォッチポイント...	ウォッチポイントを設定する		2-7の5
	D/ウォッチの削除...	ウォッチウィンドウ内のウォッチ式を削除する		2-7の3
	L/全ウォッチの削除	ウォッチウィンドウ内の全ウォッチ式を削除する		//
	T/トレース	トレースモードを ON/OFF する		2-7の2

メインメニュー	サブメニュー	機 能	ショート・カット・キー	解説項
D / デ バ グ	H/ヒストリ	ヒストリ機能を ON/OFF する		2-7の8
	B/ブレークポイント	カーソル位置にブレークポイントを設定/解除する	F・9	2-7の4
	C/全ブレークポイントの解放	すべてのブレークポイントを解除する		//
	S/カレントステートメントの設定	次に実行するステートメントをカーソル位置に移す		2-7の7
C / 関 数		現在呼び出されているプロシージャを表示し, そのプロシージャを呼び出しているプロシージャ名の位置にカーソルを移す		2-7の9
H / ヘル プ	G/全般...	全般のヘルプ情報を表示する	F・1	2-8の2
	T/キーワード	BASIC のキーワード(予約語)についてのヘルプ情報を表示する	SHIFT + F・1	2-8の1
	C/ヘルプをとじる	ヘルプウィンドウを閉じる	ESC	2-8の2

3 ショート・カット・キー

Quick BASIC のショート・カット・キーの一覧を以下に示します. テキストの編集に関するものは2-5を参照してください.

表2-3 ショート・カット・キー

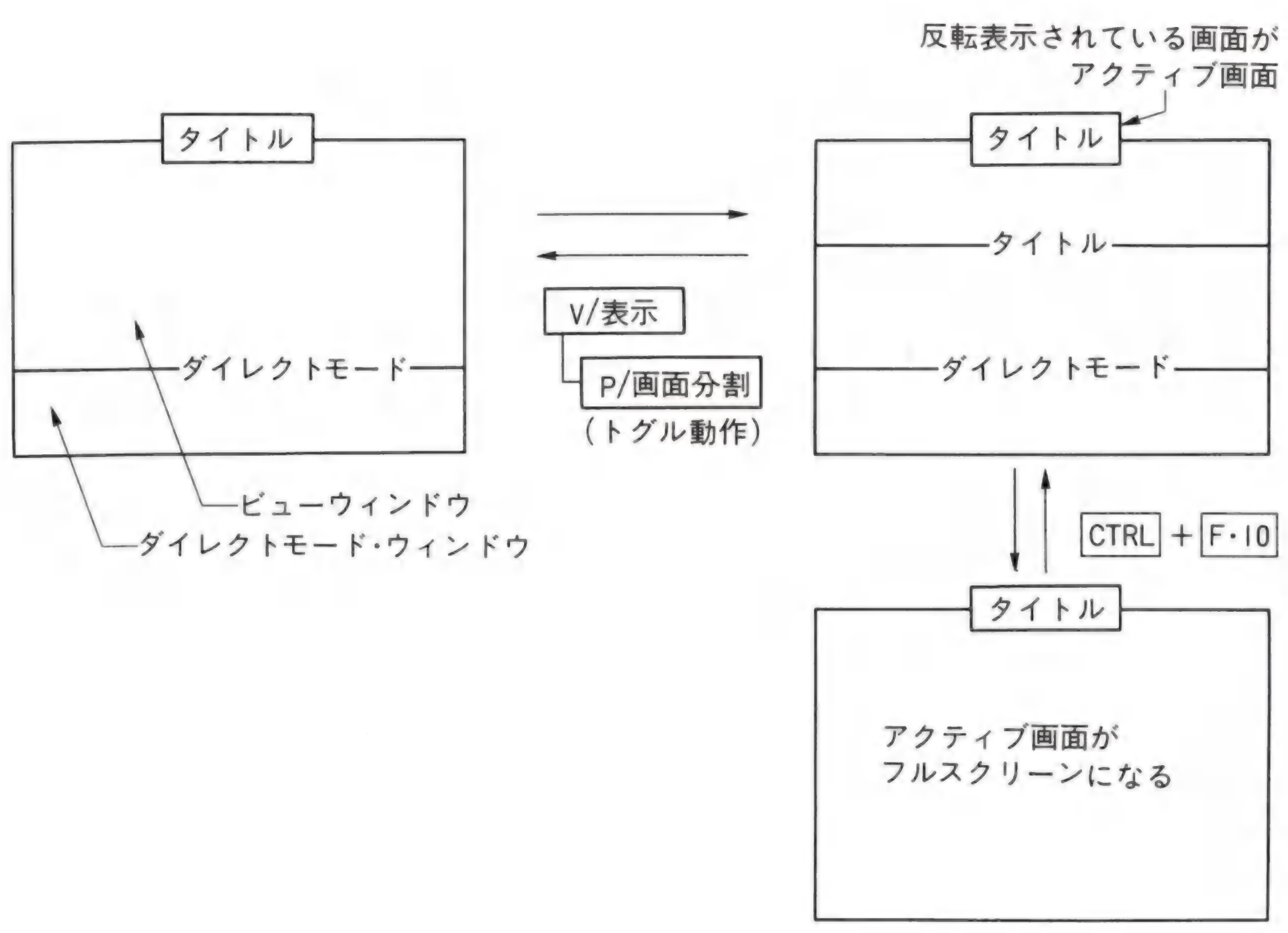
メニュー操作キー	機能
F・1	全般のヘルプ情報を表示する
F・2	SUB モジュールの一覧を表示する
F・4	QB 画面とプログラム出力画面を切り換える
F・5	STOP 文などにより中断されていたプログラムを続行する
F・6	アクティブ・ウィンドウを次の画面(下方の画面)に移す
F・7	現在のカーソル位置までプログラムを実行する
F・8	1 行トレース. プロシージャの内部もトレースする
F・9	カーソル行に, ブレークポイントを設定/解除する
F・10	1 行トレース. プロシージャの内部はトレースしない
SHIFT + F・1	指定された予約語(キーワード)についてのヘルプ情報を表示する オンライン・ヘルプ
SHIFT + F・2	アクティブ・ウィンドウに次の SUB プロシージャを表示する
SHIFT + F・5	プログラムを最初から実行する
SHIFT + F・6	アクティブ・ウィンドウを前の画面(上方の画面)に移す
SHIFT + F・8	実行された最後の20行をプログラム先頭方向にたどる
SHIFT + F・10	実行された最後の20行をプログラム終末方向にたどる
CTRL + F・2	アクティブ・ウィンドウに前の SUB プロシージャを表示する
CTRL + F・5	フルスクリーンを以前の分割画面に戻す
CTRL + F・10	フルスクリーンと分割画面を切り換えるトグル動作

2-3 ウィンドウ操作

1 ウィンドウの分割

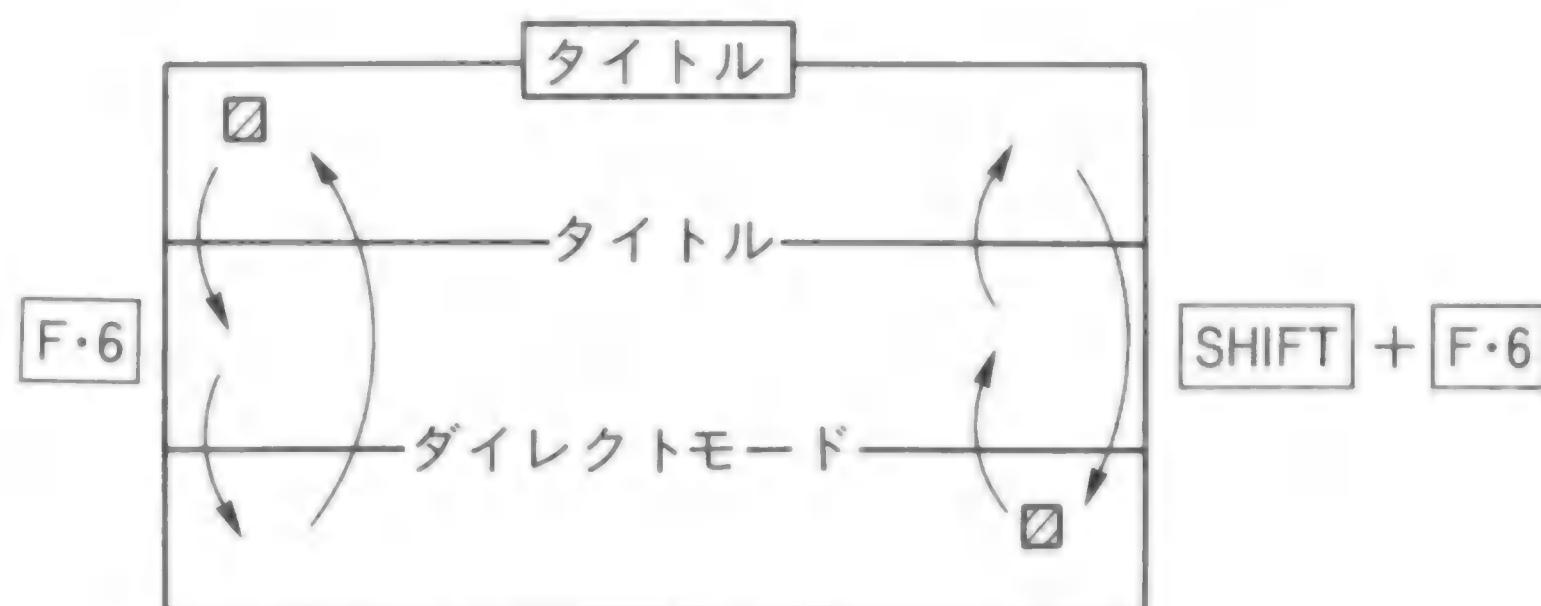
ウィンドウは、ビューウィンドウ、ダイレクトモード・ウィンドウ、ウォッチ・ウィンドウに分かれます。このうちビューウィンドウは、**V/表示**—**P/画面分割**メニューで2つに分割することができます。

また、**CTRL**+**F・10**によりアクティブ画面をフルスクリーンにしたり、元に戻したりすることができます。



2 アクティブ・ウィンドウの移動

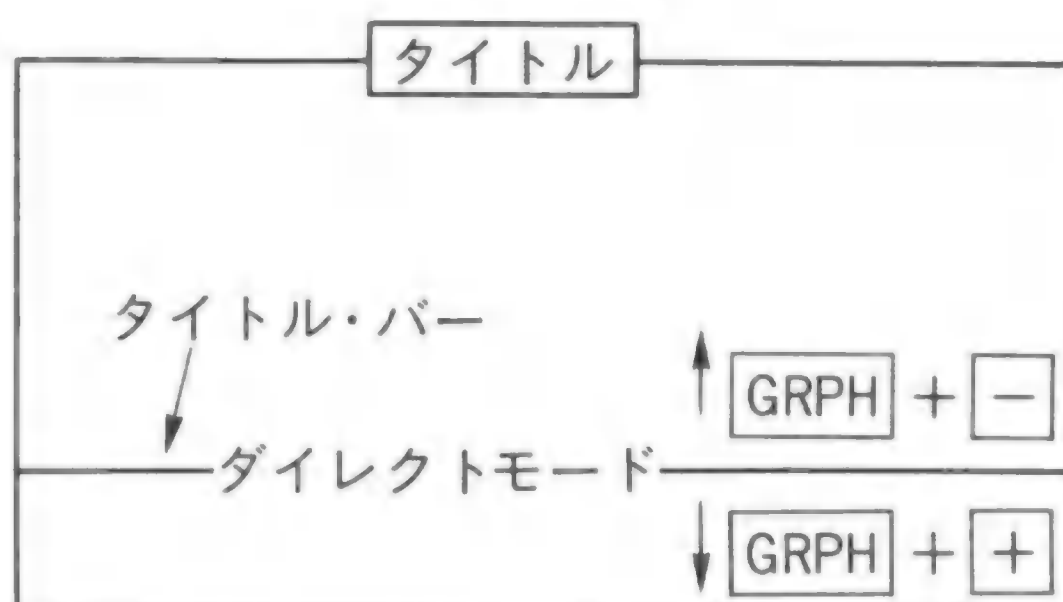
F・6, **SHIFT** + **F・6**によりアクティブ・ウィンドウを移動することができます。



マウスで移動する場合は、マウス・カーソルを移動したいウィンドウに移し、左ボタンをクリックします。

3 ウィンドウの拡大縮小

アクティブ・ウィンドウのサイズを拡大または縮小するには、**GRPH** + **+**, **GRPH** + **-**を用います。



マウスを用いるならタイトル・バーにマウス・カーソルを合わせ、左ボタンを押しながらマウスを上/下に移動します。これで移動方向にウィンドウが拡大/縮小します。

4 実行画面への切り換え

プログラムの実行結果が出力されている画面と QB 画面との切り換えは、**F・4**または **V/表示** - **U/実行画面**で行います。



2-4

ファイル操作

1

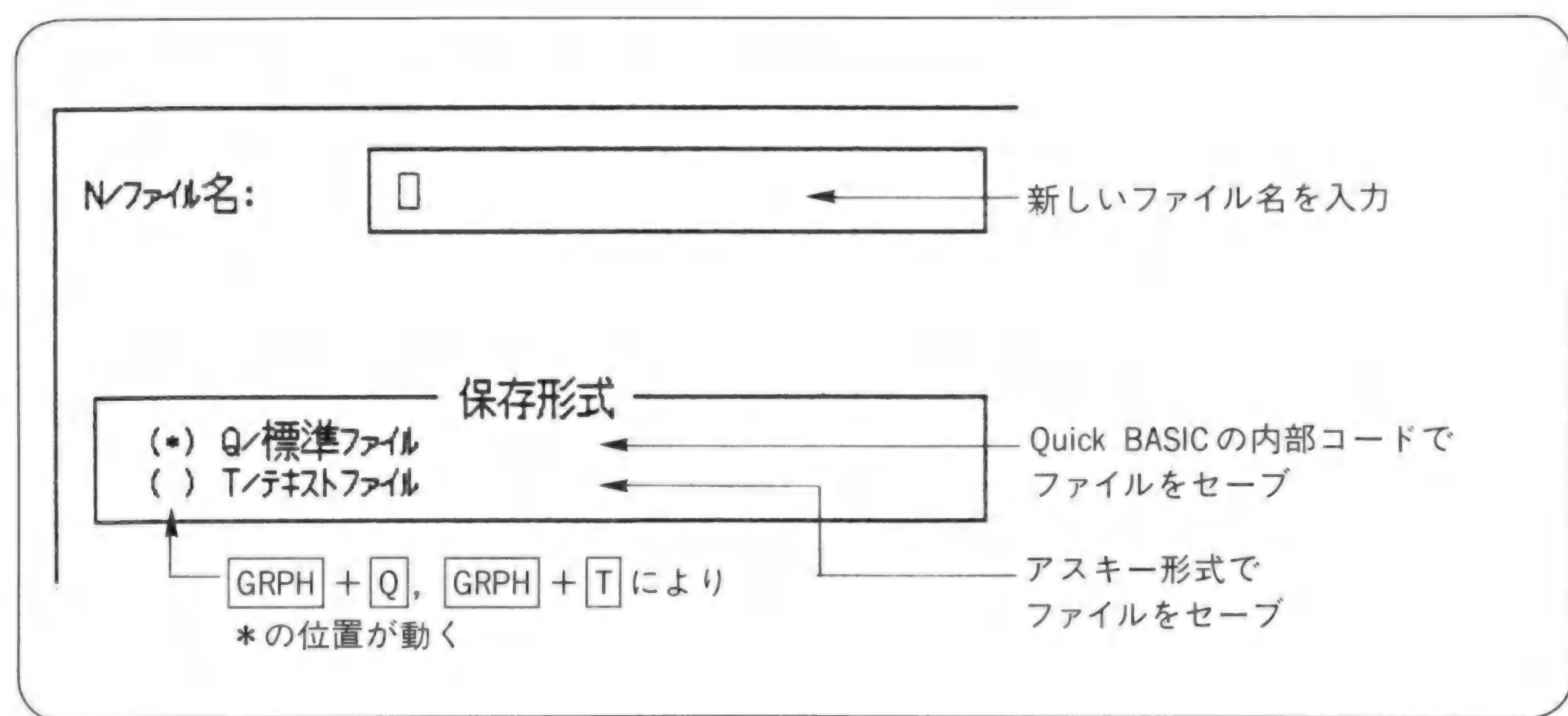
プログラムのセーブ

エディタ上で作ったファイルを保存するには、**F/ファイル**メニューの中の、次の3通りの方法があります。

- S/保存
- …現在のアクティブ・ウィンドウに表示されているファイルをセーブします。
- V/すべて保存
- …サブモジュールも含め、すべてをセーブします。マルチモジュールで作業しているときに便利です。
- A/名前を変えて保存...
- …タイトル・バーに表示されている名前と異なる名前で保存します。

単一モジュールのプログラムの場合は**S/保存**，マルチモジュールのプログラムの場合は**V/すべて保存**を使うのが一般的です。

タイトル・バーが、「Untitled」(ファイル名がない場合)のときに**S/保存**か**V/すべて保存**を選択した場合、**A/名前を変えて保存...**を選択した場合は、次のようなダイアログ・ボックスが開くので、ファイル名およびファイルの保存形式を指定してください。



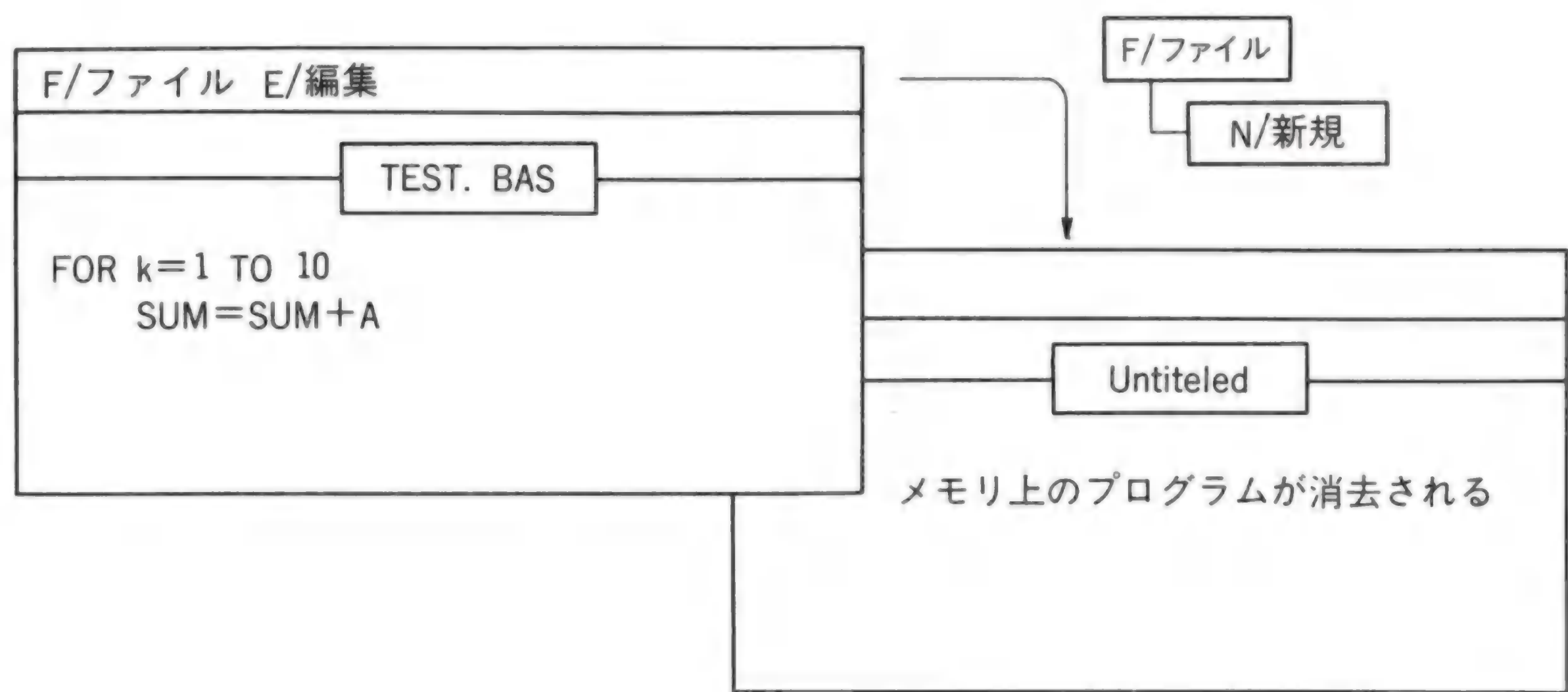
「Q/標準ファイル」の形式でセーブされたファイルはQuick BASIC の内部コードでセーブされているので、TYPE コマンドで見ることにはできませんし、ほかのエディタを用いて編集することもできません。

MS-DOS 上のソーステキストとして一般性を持たせるには、「T/テキストファイル」の形式でセーブしてください。

一度セーブされたファイルをロードし再びセーブする場合は、前の保存形式がデフォルトとなります。

2 新規プログラムを作る

現在のプログラムの開発が終わり、次のプログラムを作りたいときは、一度 Quick BASIC を終了してから再起動してもよいのですが、QB の起動には時間がかかります。一般に **F/ファイル** メニューの **N/新規サブメニュー** を用いて、現在のプログラムをメモリ上から消し、次のプログラム開発に入ります。



3 プログラムのロード

エディタ上にプログラムをロードするには、**F/ファイル**メニューの中の次の2通りの方法があります。

- O/読込...** ...現在のエディタ上のプログラムを消してから指定されたプログラムをロードします。
- M/追加...** ...現在作業中のプログラムのカーソル位置の上部に挿入します。

N/ファイル名:

.EPR

← ロードするファイル名

A:¥QB¥SOURCE

A.BAS
[.]
[-A-]
[-B-]
[-C-]
[-D-]

← カレントディレクトリにあるファイル一覧
ここにカーソルを移し ☒ で指定してもよい

4 プログラムのプリントアウト

F/ファイル—**P/印刷...**メニューにより、メモリ上のプログラムをプリンタに出力できます。次のようなダイアログ・ボックスが出るので、プリントアウトする範囲を指定してください。

印刷

() S/指定範囲

() W/アクティブウインドウ

(*) M/カレントモジュール

() A/全体

← ウィンドウ内で反転表示されているテキスト

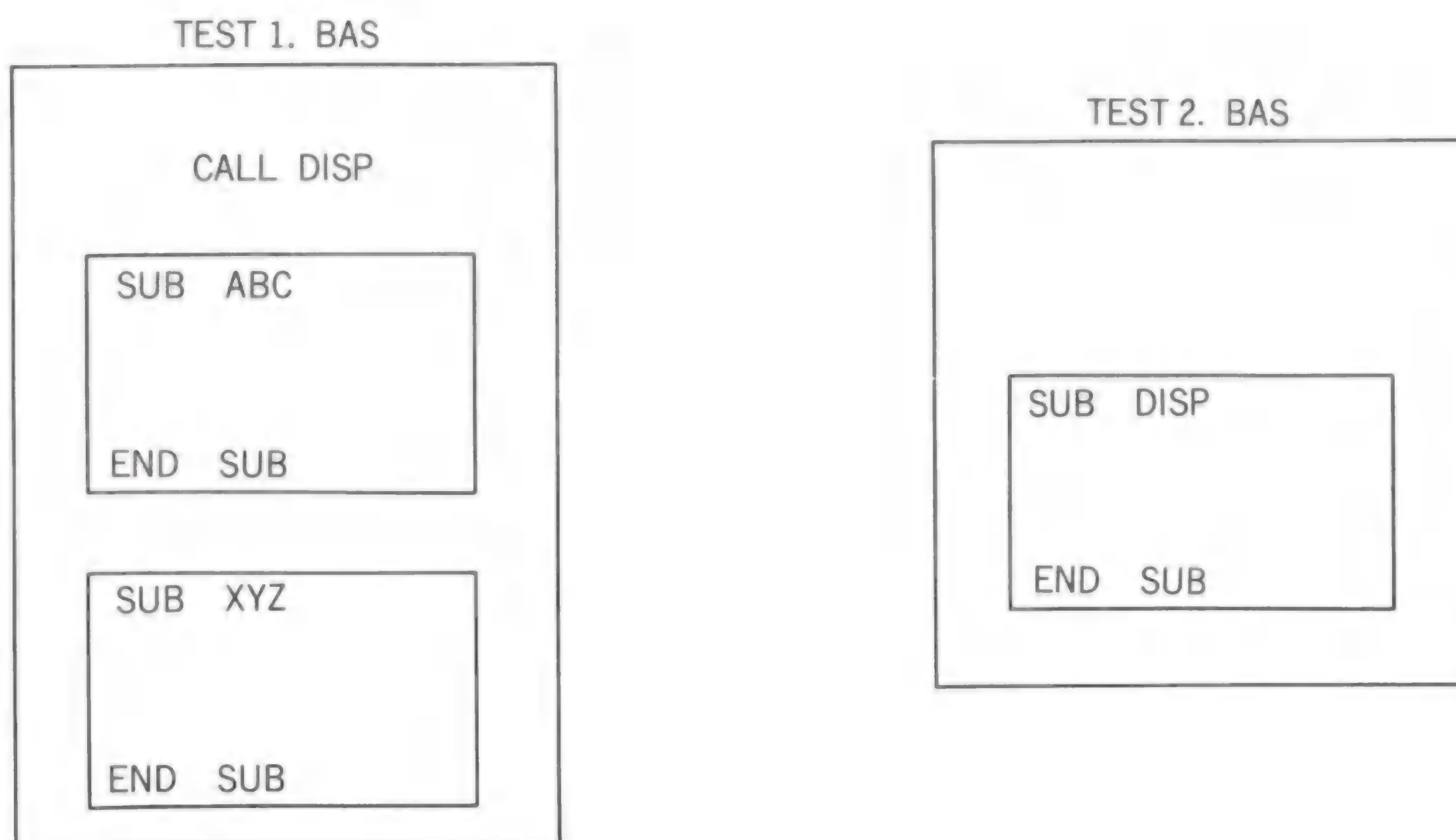
← 表示されている画面

← 作業中のモジュール

← メモリ上の全モジュール

5 モジュール管理

たとえば、次のような2つのプログラムがあったとします。

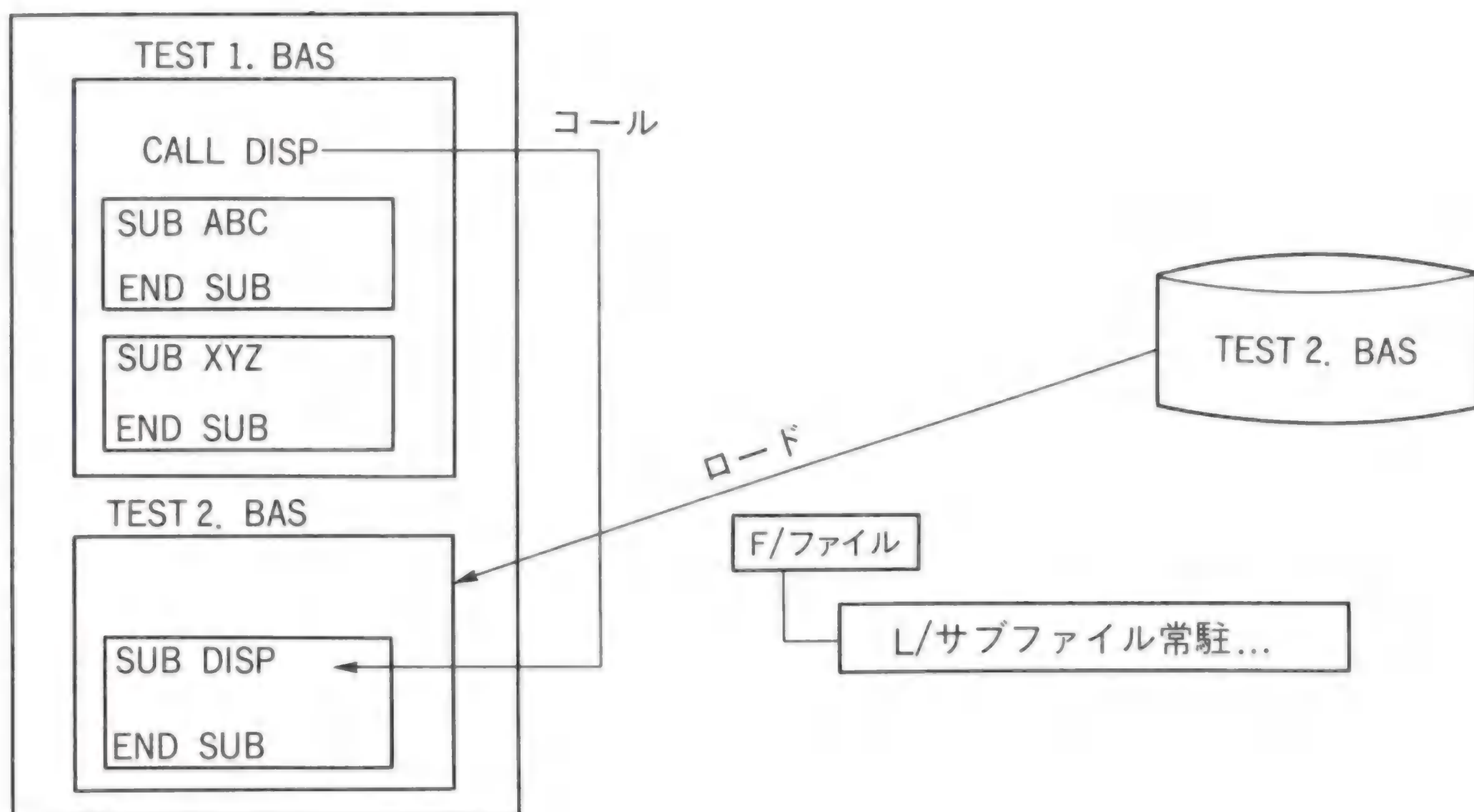


QB では、複数のプログラムをメモリ上に並存させて管理することができます。メモリ上にロードされたプログラムをモジュールと呼びます。実行が開始されるモジュールをメインモジュールと呼び、他のモジュールをサブモジュールと呼びます。

QB 上にロードされているモジュールの各プロシージャ(SUB~END SUB, FUNCTION~END FUNCTIONで定義されている)は、他のプログラム(モジュール)から利用することができます。

■サブファイルのロード

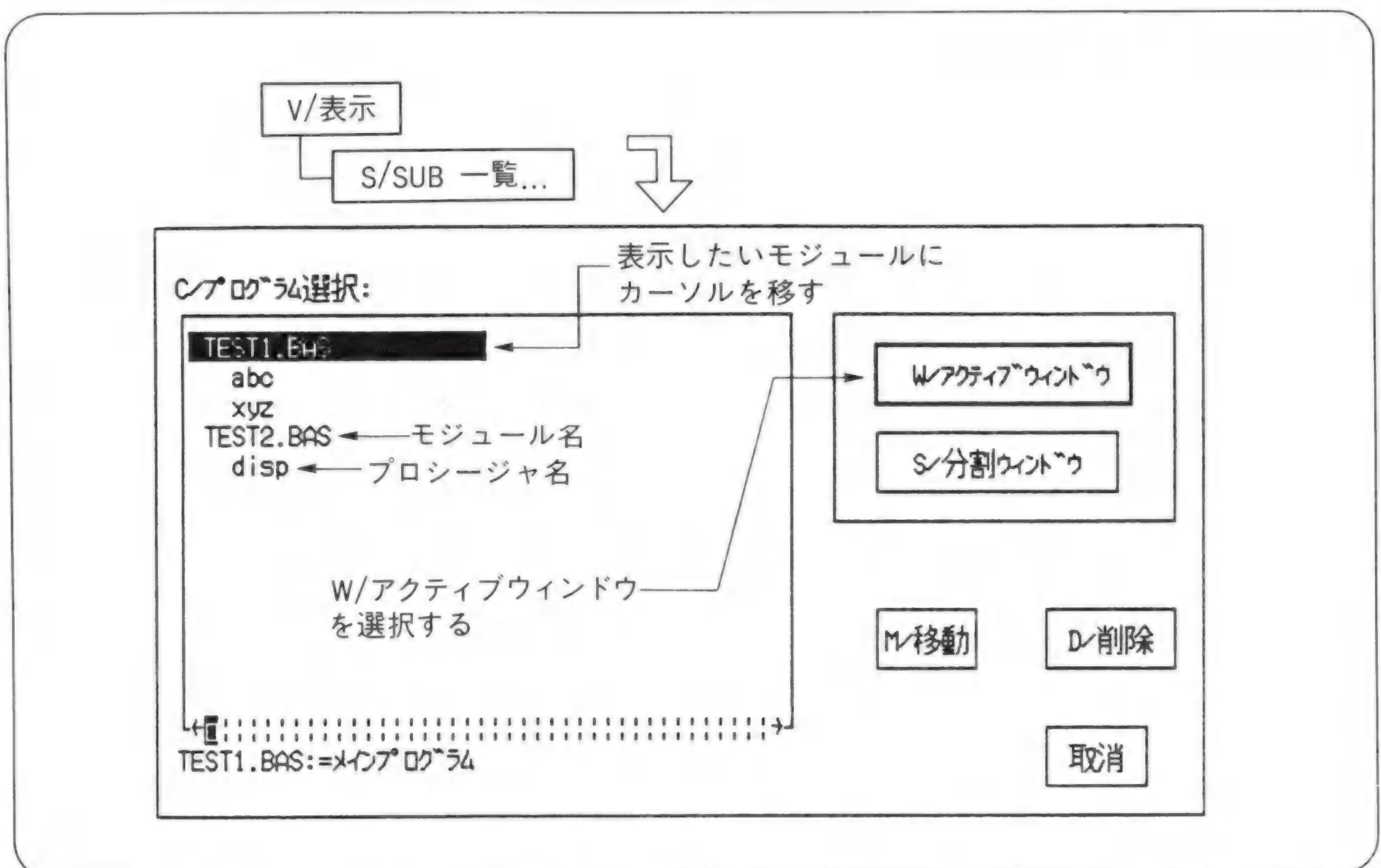
現在 TEST1.BAS を QB 上で編集しているものとし、TEST2.BAS という別のファイルを QB 上にロードするには、**F/ファイル**—**L/サブファイル常駐...**を選択します。



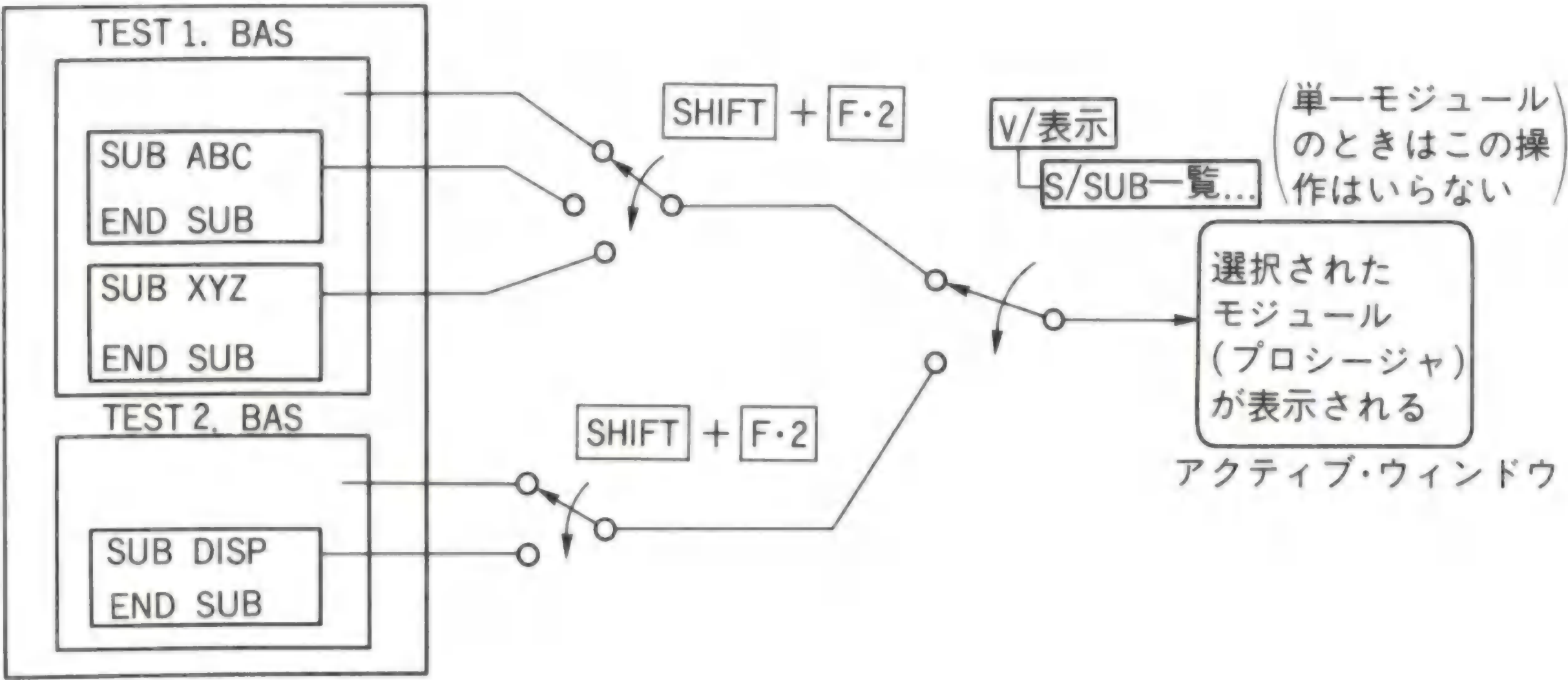
■モジュール画面の切り換え表示

QB 上のモジュールおよびプロシージャはそれぞれ独立に管理されているため、同じ画面に同時に表示することはできません。

アクティブ・ウィンドウにどのモジュール(プロシージャ)を表示するかは、**V/表示**—**S/SUB 一覧...**または**F・2**を用いて次のように行います。

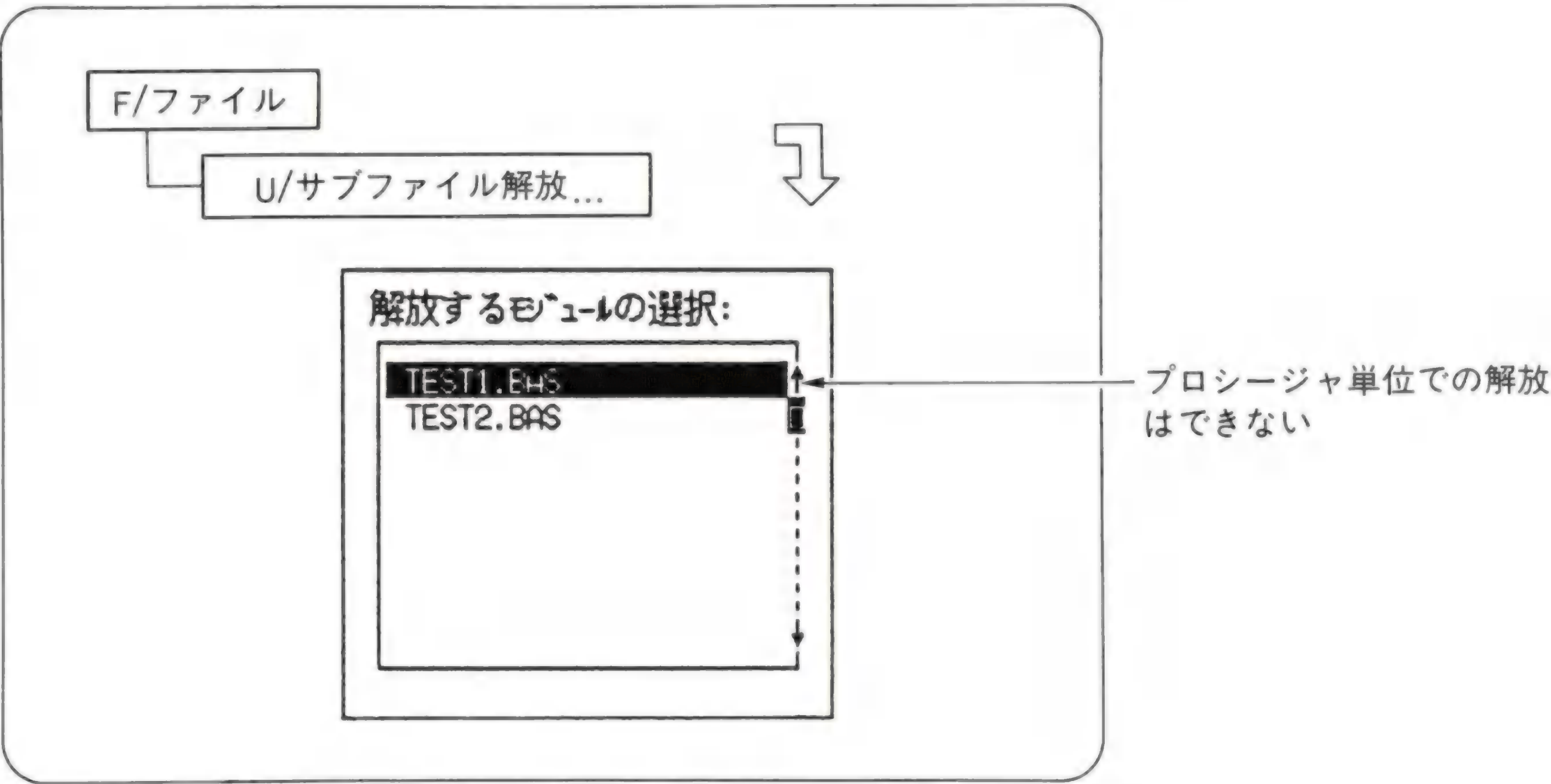


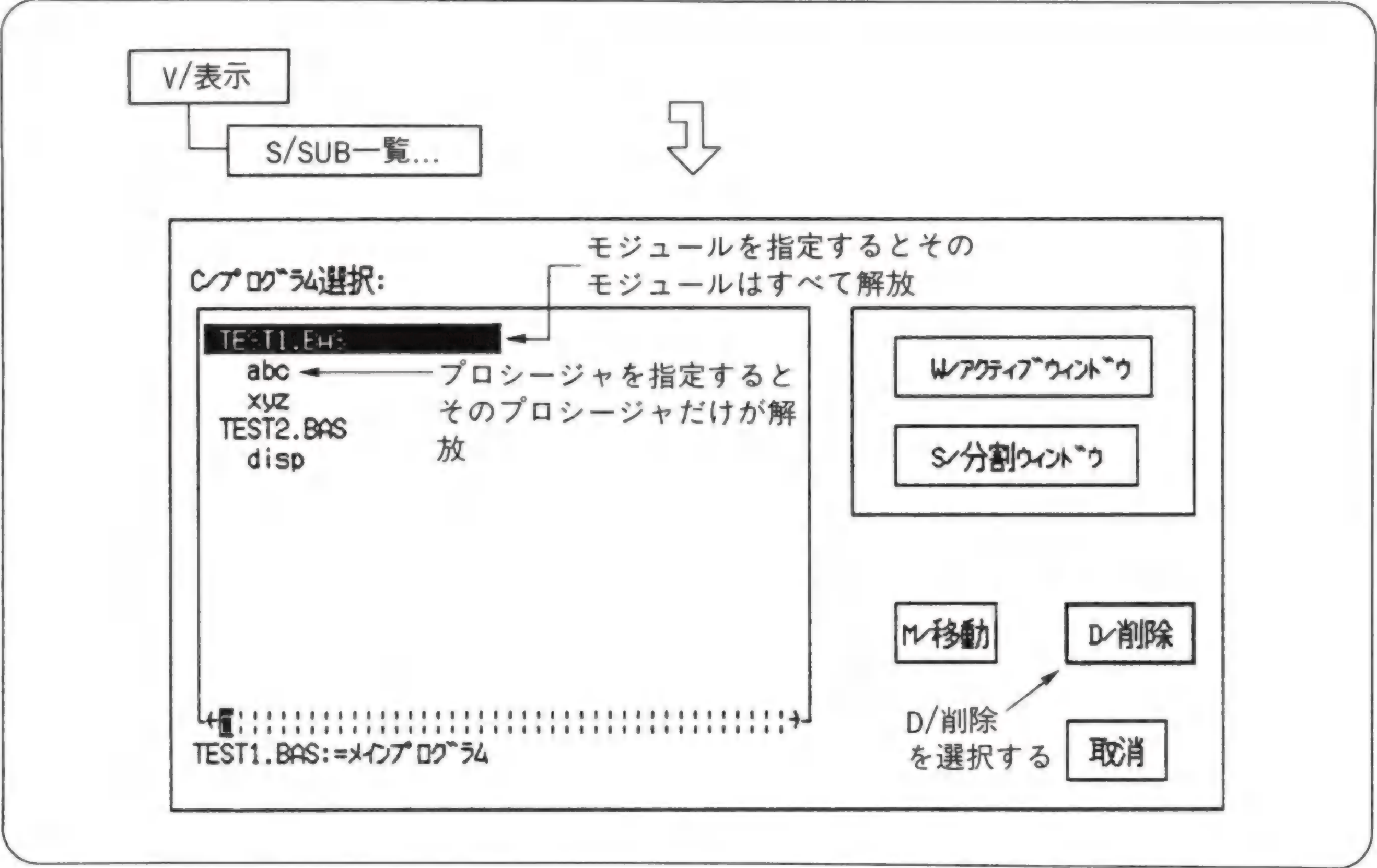
なお、**[SHIFT] + [F・2]**を用いれば、**[V/表示] — [S/SUB 一覧...]**でアクティブになっているモジュール内の各プロシージャをアクティブ・ウィンドウに切り換え表示できます。



■モジュールの解放

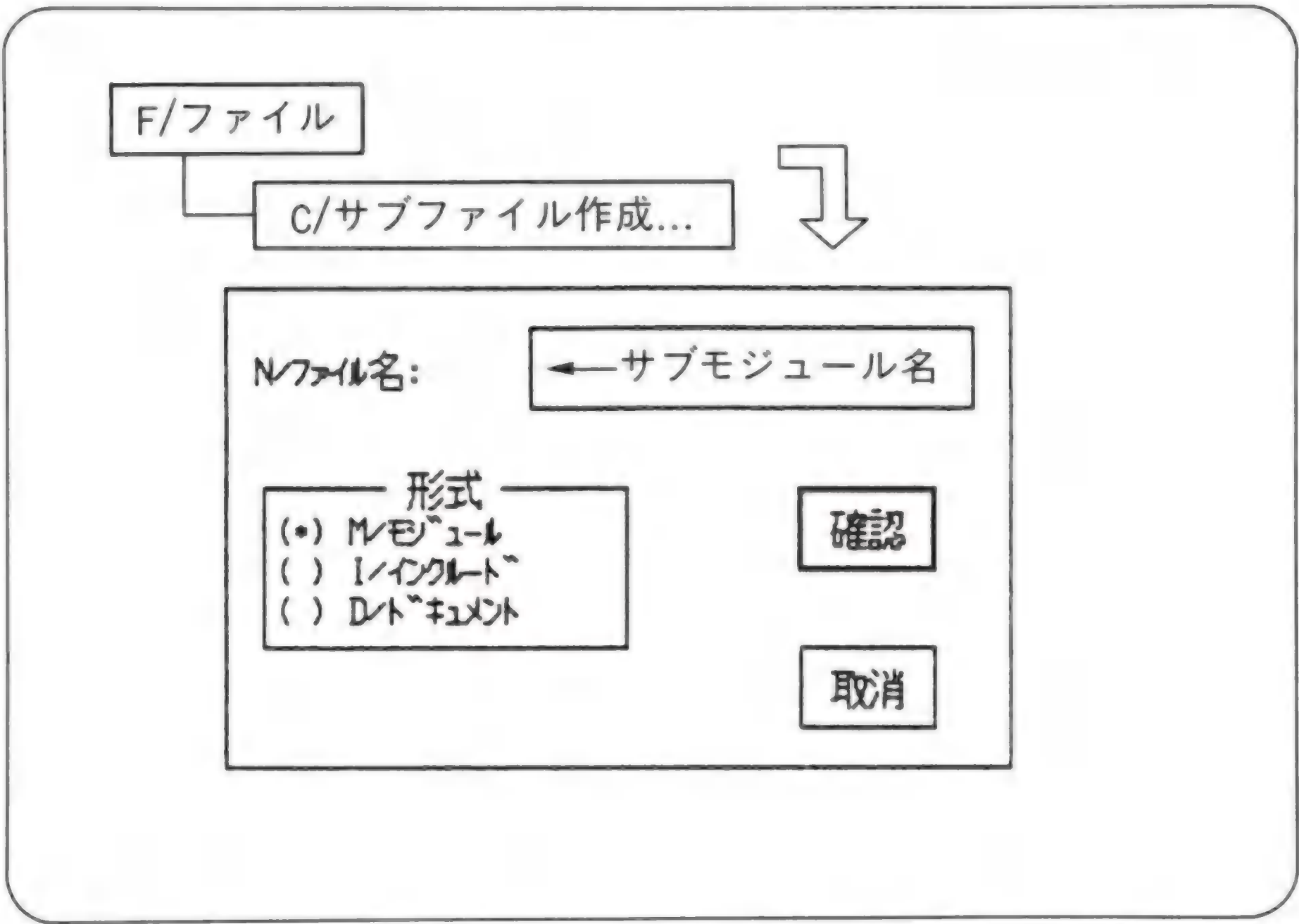
QB 上にロードされているモジュールを削除する場合には、**[F/ファイル] — [U/サブファイル解放...]**または、**[V/表示] — [S/SUB 一覧...] の [D/削除]**を用います。





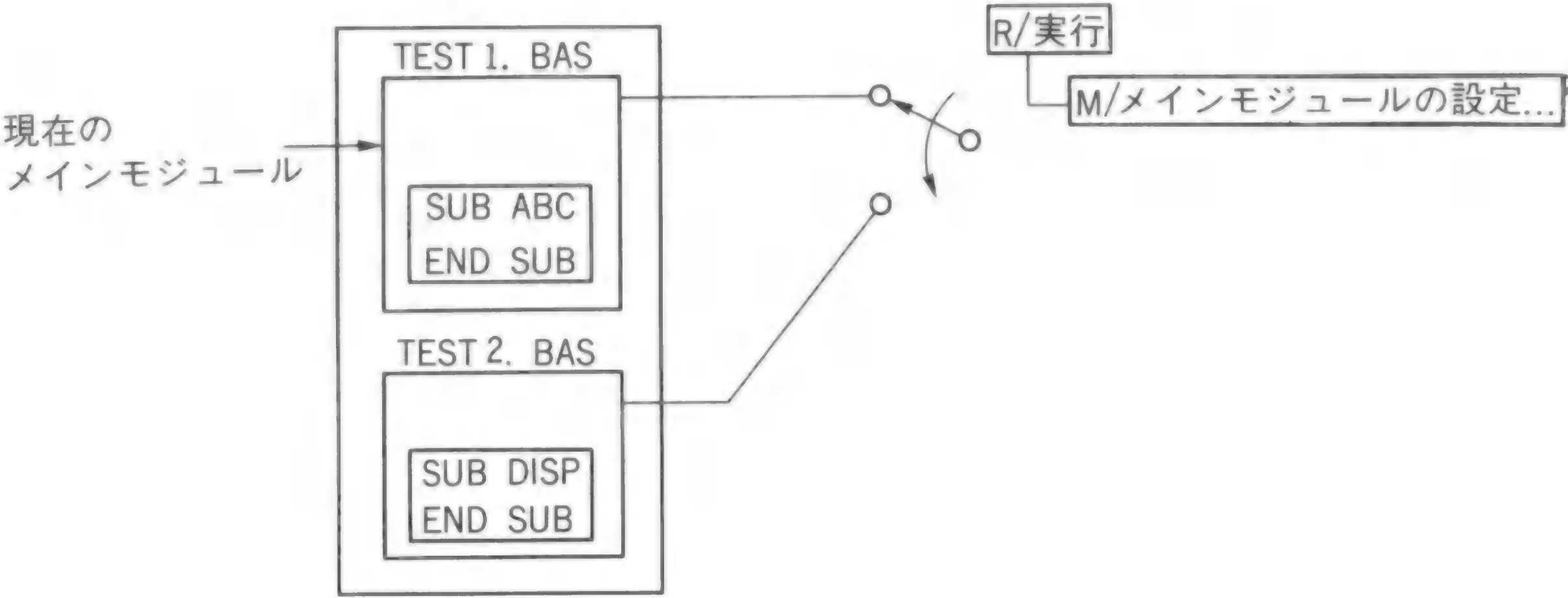
■サブモジュールの作成

現在メモリ上にあるプログラムに新しくサブモジュールを加えるには、**F/ファイル**—**C/サブファイル作成...**を用います。



■メインモジュールの設定

QB 上に複数のモジュールがロードされている場合、実行を開始するモジュール (メインモジュール) を **R/実行** — **M/メインモジュールの設定...** により切り換えることができます。



6 ドキュメント・ファイルの作成

Quick BASIC のエディタで、CONFIG.SYS や AUTOEXEC.BAT などのテキスト・ファイル(ドキュメント・ファイル)を作る場合、デフォルトモードでは構文チェック機能などが働くためうまくいきません。

テキスト・ファイルの作成法を以下に示します。

- ① **F/ファイル**—**N/新規**でメモリ上のプログラムを消去する。
- ② **F/ファイル**—**C/サブファイル作成...**を選択し、次のダイアログ・ボックス内の各項目を設定する。

N/ファイル名: CONFIG. SYS

作成するテキストの名前を入力

形式

() M/モジュール
() I/インクルード
(*) D/ドキュメント

GRPH + D でドキュメントを指定

- ③ テキストを入力する。

CONFIG. SYS

テキストを入力する

- ④ **F/ファイル**—**S/保存**でテキストをセーブする。

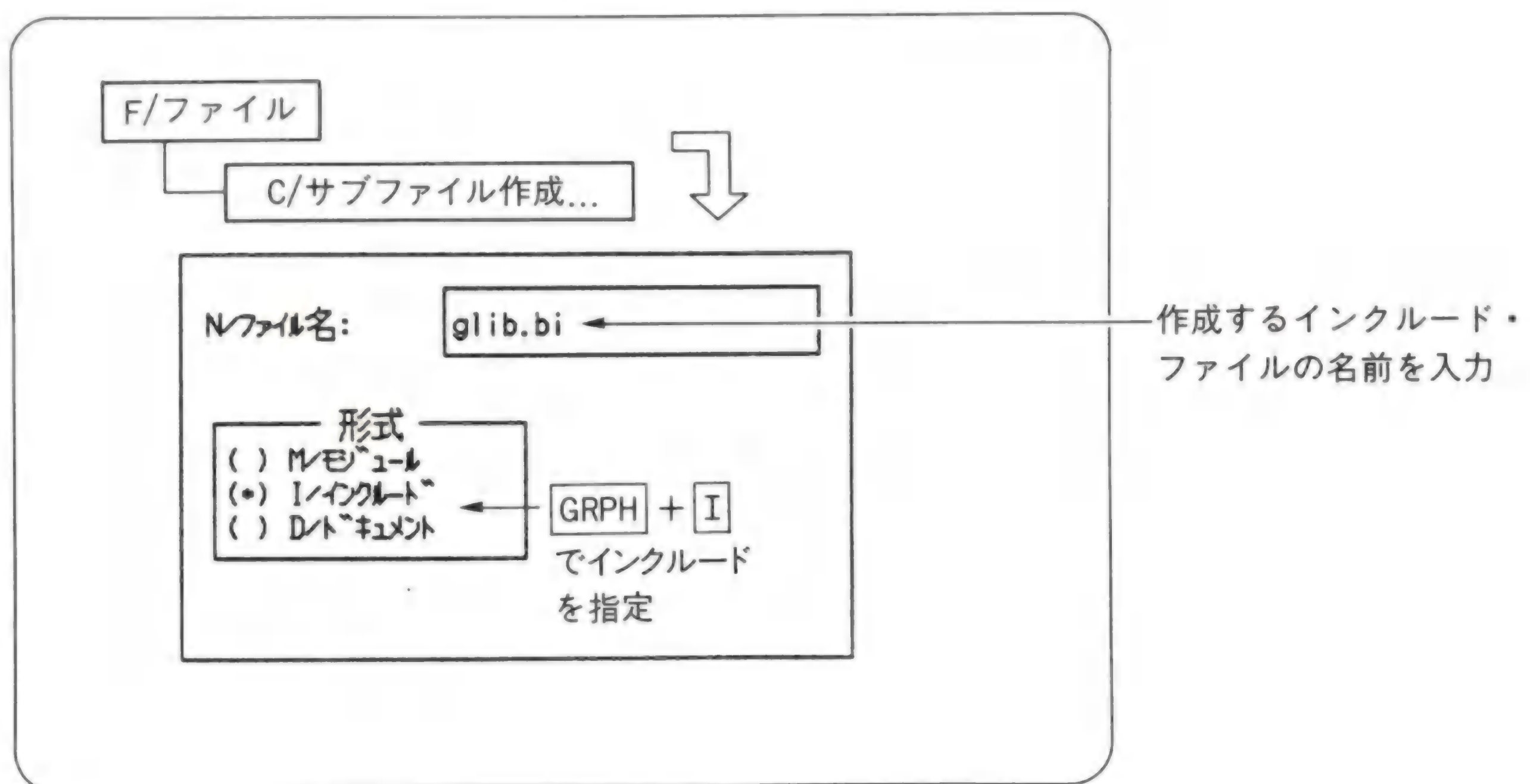
なお、すでに作成してあるテキスト・ファイル(ドキュメント・ファイル)を扱う場合は、上の②の代わりに**F/ファイル**—**L/サブファイル常駐...**を選択します。

7 インクルード・ファイル

インクルード・ファイルは、メタコマンドの「\$INCLUDE: 'ファイル名」で指定されるテキスト・ファイルです。

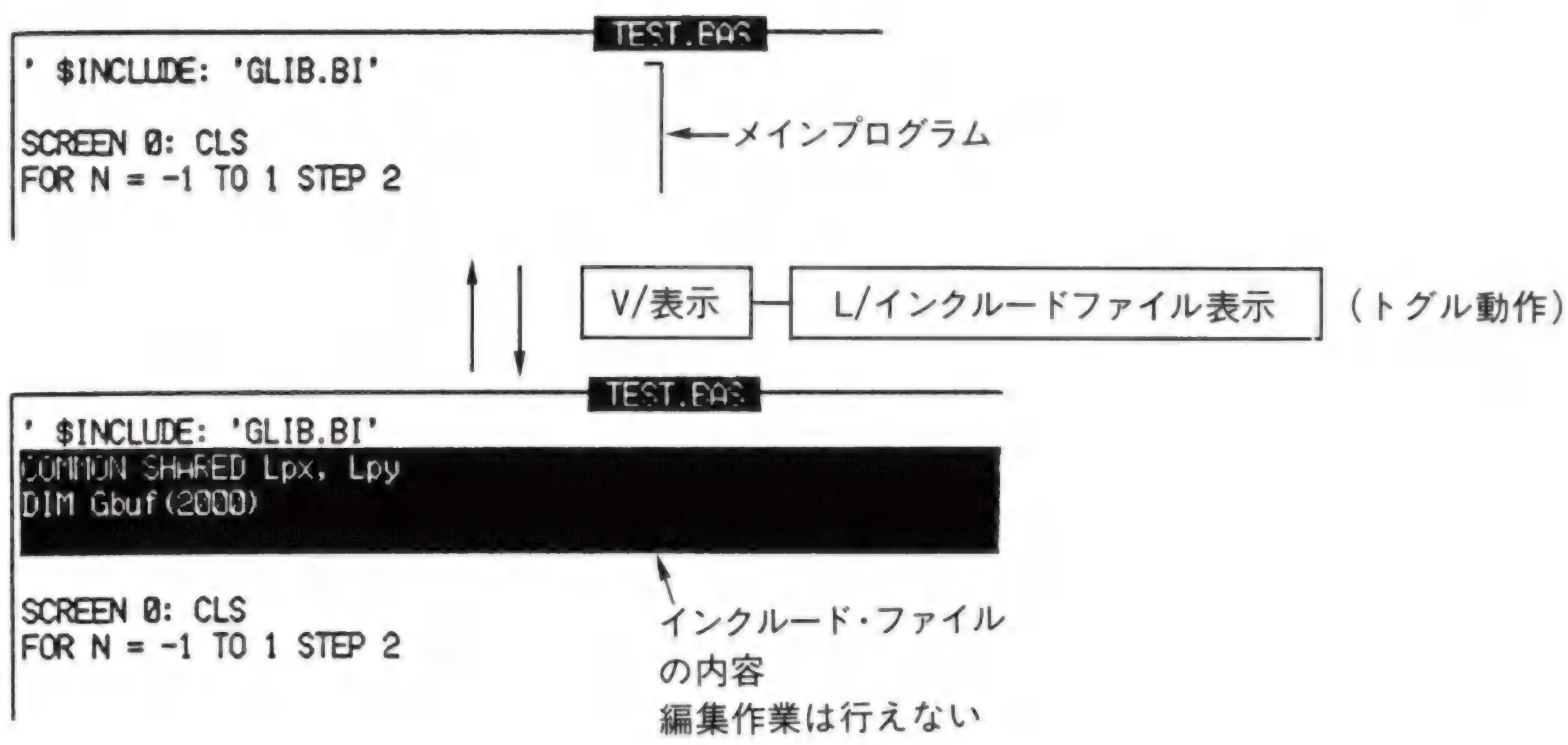
■ インクルード・ファイルの作成

インクルード・ファイルを作成するのは、ドキュメント・ファイルの作成方法と同様ですが、**F/ファイル**—**C/サブファイル作成...**を選択し、「I/インクルード」形式を指定します。



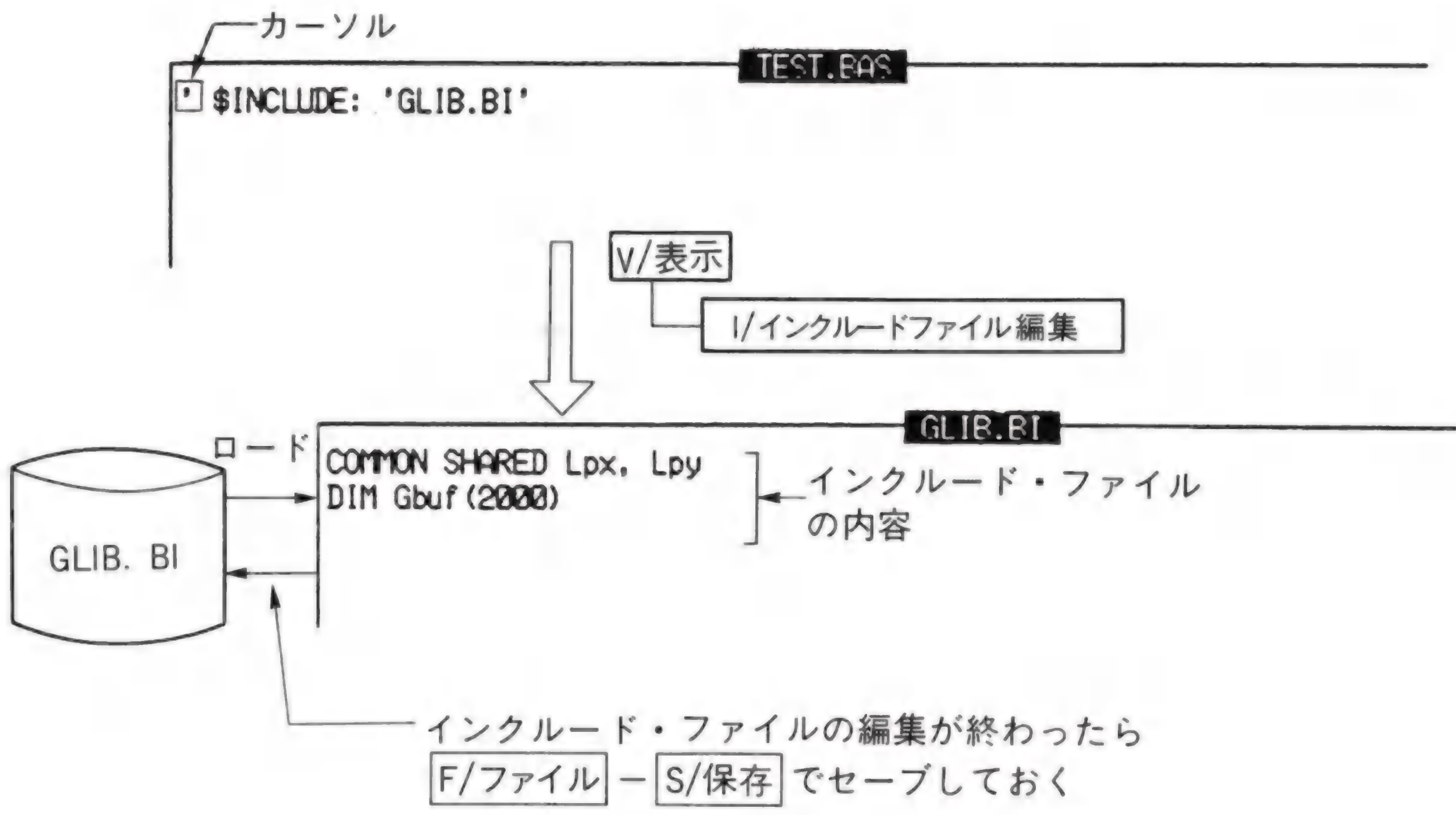
■ インクルード・ファイルの表示

メインプログラム中の\$INCLUDEで指定されているインクルード・ファイルの内容を表示するには、**V/表示**—**L/インクルードファイル表示**を選択します。逆に表示を消す場合にもこのメニューを選択します。



■インクルード・ファイルの編集

カーソルを\$INCLUDEメタコマンドのある行に置いて、**V/表示** — **I/インクルードファイル編集** を選択します。



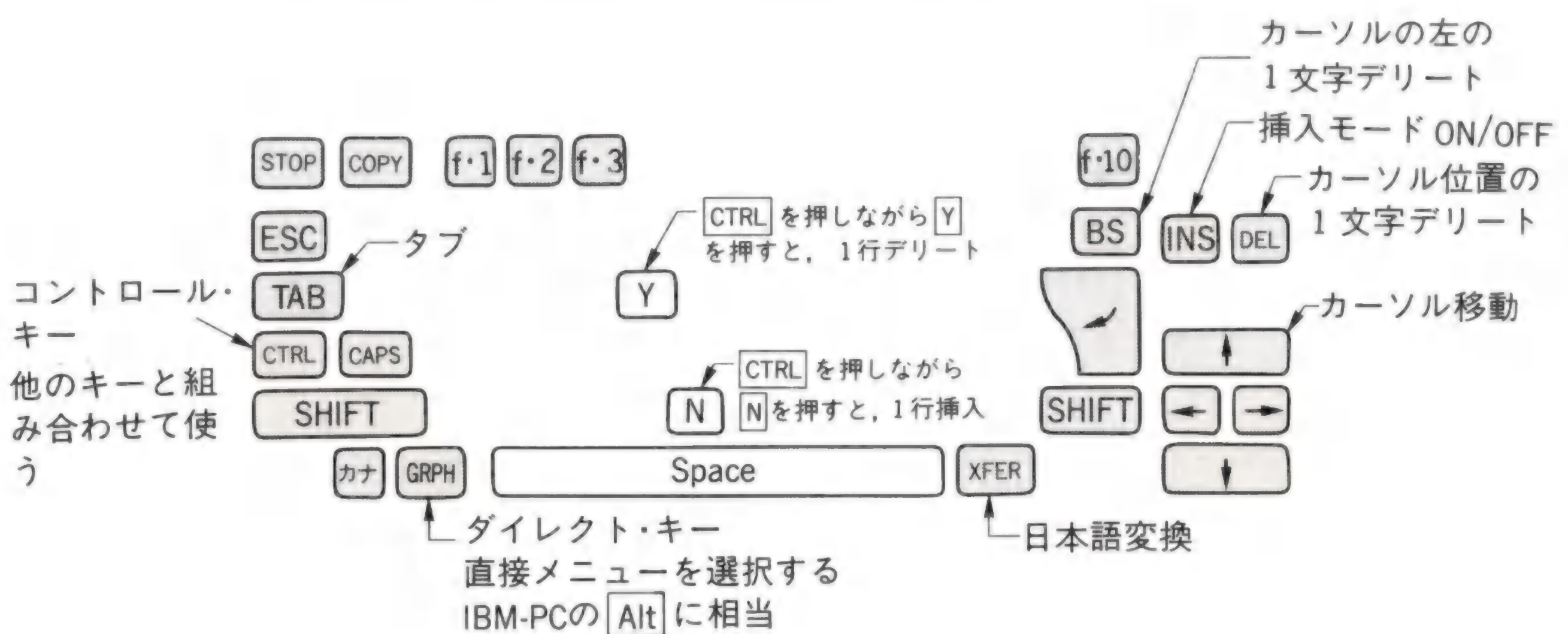
なお、すでにインクルード・ファイルがメモリに在駐している場合は、**V/表示** — **S/SUB一覧...** によりインクルード・ファイルをアクティブウィンドウに表示して、編集作業に入ってください。

2-5

編集(エディタ)

1 キー配置


エディタ上で使う重要なキーの配置を以下に示します。



2 編集コマンド一覧

テキスト編集で使うキー・コマンドの一覧を以下に示します。

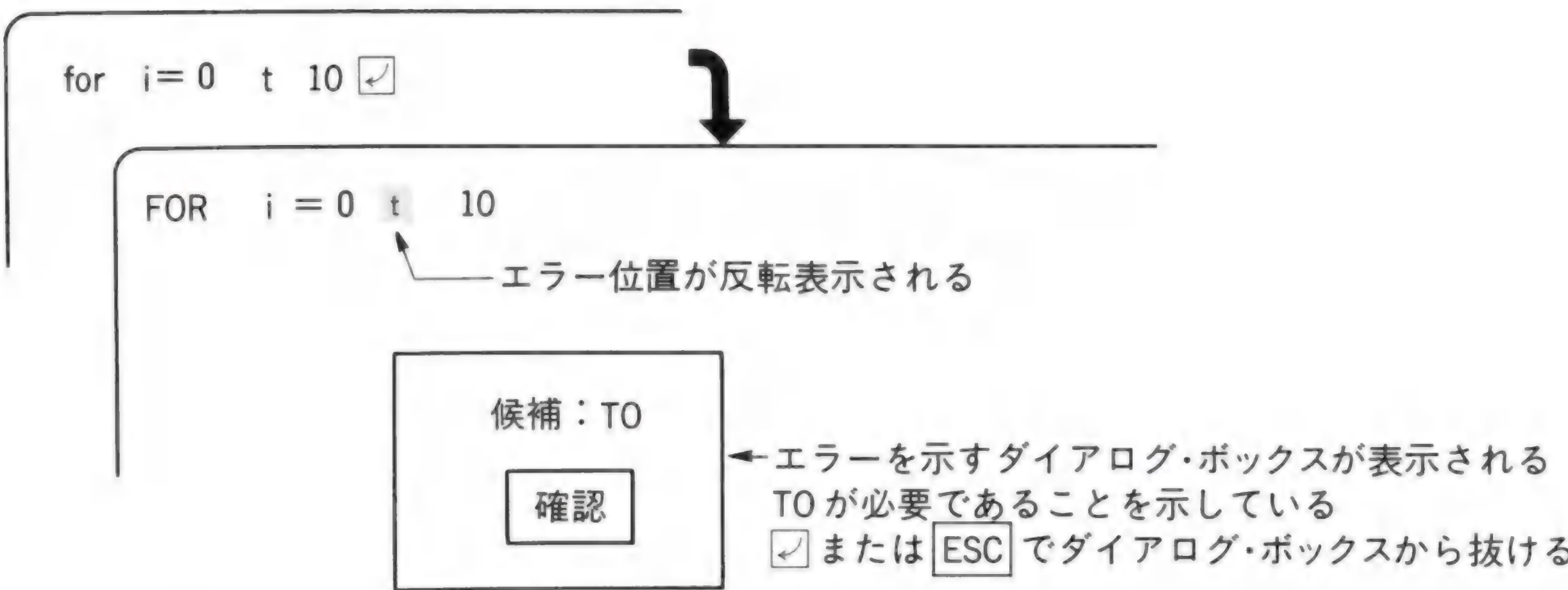
表 2-4 キー・コマンド一覧

	編集キー	機能
カーソルの移動		カーソルを左に1文字移動
		カーソルを右に1文字移動
		カーソルを上1行移動
		カーソルを下1行移動
	 + 	左に1単語分移動
	 + 	右に1単語分移動
		カレント行の最初の文字に移動
	 +  	カレント行の物理的な左端に移動
	 + 	次の行の最初の文字に移動
		カレント行の最後の文字に移動
	 +  	ウィンドウの上端に移動
	 +  	ウィンドウの下端に移動
	 +  	モジュール/プロシージャの先頭に移動
	 +  	モジュール/プロシージャの終わりに移動
挿入		挿入モードのON/OFF
	 + 	カレント行の上に1行挿入。カーソルが行の先頭位置にないとその行が中断されるので、  でカーソルを先頭文字に移してから  +  を行う
		カレント行の下に1行挿入。カーソルが行の終末位置にないとその行が中断されるので、その場合は  でカーソルを行の終末文字に移してから  を行う

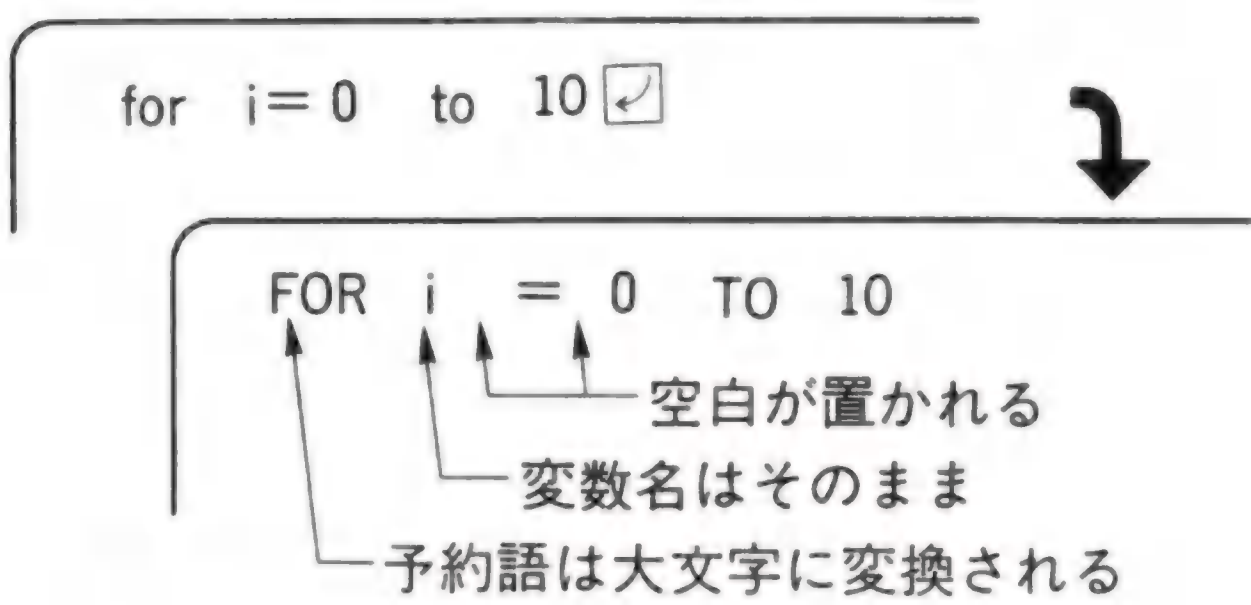
	編集キー	機能
削除	<div>CTRL</div> + <div>Y</div>	カーソル行を削除(この内容はクリップボードに保存される)
	<div>CTRL</div> + <div>Q</div> <div>Y</div>	カーソル位置から行末までを削除(この内容はクリップボードに保存される)
	<div>BS</div>	カーソル位置の左の文字を削除
	<div>DEL</div>	カーソル位置の文字を削除
	<div>CTRL</div> + <div>T</div>	1 単語を削除
クリップボード	<div>SHIFT</div> + <div>INS</div>	クリップボードの内容をカーソル位置にコピー
	<div>SHIFT</div> + <div>DEL</div>	指定した(反転表示された)テキストを削除し, クリップボードにセーブ
	<div>CTRL</div> + <div>INS</div>	指定した(反転表示された)テキストを削除せず, そのままにしてクリップボードにセーブ
タブ	<div>TAB</div>	次のタブ位置に移動
	<div>SHIFT</div> + <div>BS</div>	バック・タブ(1 つ前のタブ位置に戻る)
スクロール	<div>CTRL</div> + <div>W</div> <div>CTRL</div> + <div>Z</div> <div>ROLL DOWN</div> <div>ROLL UP</div> <div>CTRL</div> + <div>ROLL UP</div> <div>CTRL</div> + <div>ROLL DOWN</div>	上に 1 行分スクロール 下に 1 行分スクロール 上に 1 画面分スクロール 下に 1 画面分スクロール 左に 1 画面分スクロール 右に 1 画面分スクロール
テキストの指定	<div>SHIFT</div> +カーソルの移動 コマンド・キー	<div>SHIFT</div> を押しながらカーソルの移動コマンド・キー(<div>←</div> ~ <div>CTRL</div> + <div>Q</div> <div>C</div>)を押すと, カーソルが移動した範囲が反転表示され, テキストが指定されたことになる
サーチ	<div>F・3</div>	次に一致する検索文字列を探す
	<div>CTRL</div> + <div>¥</div>	指定したテキストを探す
	<div>CTRL</div> + <div>Q</div> <div>A</div>	指定したテキストを探し, 新しいテキストで置き換える

3 自動構文チェック機能

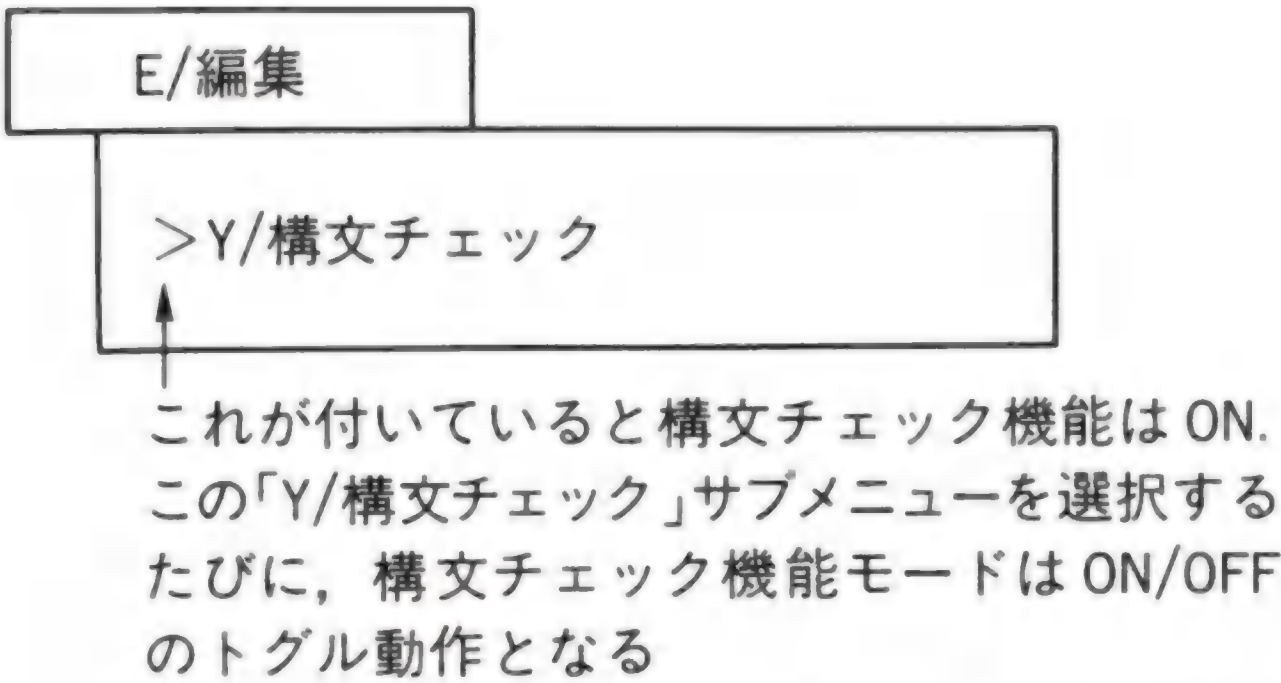
Quick BASICのエディタは、自動構文チェック機能が付いています。☐または☐を押すと、その行の内容が正しいBASIC文法にしたがっているかチェックし、もし誤っていればメッセージを出してくれます。



1行の内容が正しければ、予約語(for, ifなど)を大文字に変換し、演算子の両脇に空白を補います。また、インタプリタがその行の内容をBASICの内部コードに変換して、メモリに格納します。



Quick BASICのエディタには自動構文チェックが付いているため、その行に誤りがあればエラーメッセージが表示され、入力作業を中断してしまいます。もし、このようないねいなサポートがきらいなら、自動構文チェック機能をOFFにしておけばよいでしょう。



注) Ver. 4.5では、☐オプションの☐Y/構文チェックを指定します。また>ではなく*という表示になりました。

4 オート・インデント機能

プログラムを見やすくするために空白やタブを適当に入れ、while や for などのループ構造や、if then else などのブロックがひと目でわかるように、書き出す先頭位置を右に下げます。これをインデント(字下げ)といいます。

インデントの間隔は何文字でもかまいませんが、通常は TAB の間隔でとっていきます。Quick BASIC の TAB 間隔は 4 文字です([V/表示]メニューの [O/オプション] サブメニューで変更できます)。

注) Ver.4.5では、[O/オプション]の[D/画面設定]を指定します。

```
FOR i=1 TO 10
  SUM=0
  FOR j=1 TO N
    SUM=SUM+A
    TAB
    4文字
    TAB
    4文字
    3
  NEXT j
NEXT i
```

オート・インデント機能とは、次のように $\boxed{\hookrightarrow}$ で次の行に移ったときに、直前の行の先頭文字位置に自動的に頭合わせをすることをいいます。

```
FOR i=1 TO 10 $\boxed{\hookrightarrow}$ 
  TAB FOR j=1 TO N $\boxed{\hookrightarrow}$ 
```

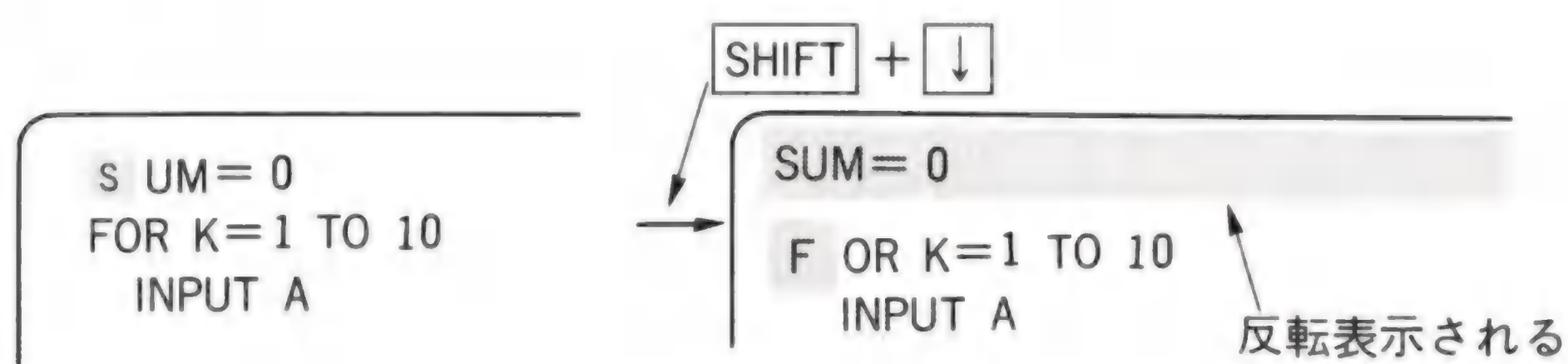


↑ 直前の行の先頭文字位置にカーソルがくる(オート・インデント)

5 テキストの指定

クリップボードを介したテキストのムーブ/コピーでは、対象となるブロック(テキスト)を指定しなければなりません。**[SHIFT]**+カーソルの移動コマンド・キー(**[←]**~**[CTRL]**+**[Q]****[C]**)を押すと、カーソルが移動した範囲が反転表示され、テキストが指定されたことになります。

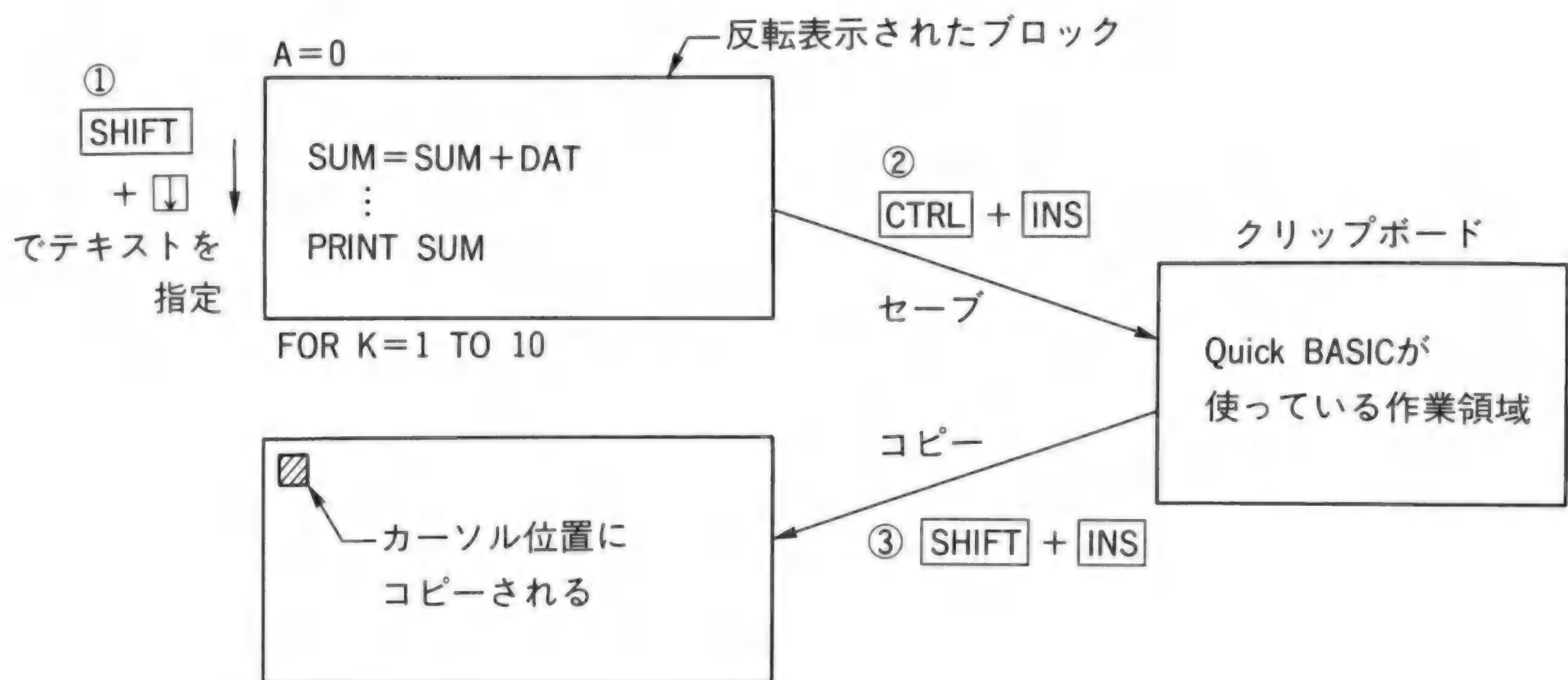
テキストが指定されている状態で**[E/編集]**—**[E/削除]**または**[DEL]**を行うと、そのテキスト全体が削除されます。



マウスを用いてテキストを指定するには、マウスの左ボタンを押しながらマウスを移動します。マウス・カーソルが移動した範囲が反転表示され、テキストを指定したことになります。

6 テキストのブロック・コピー/ムーブ

クリップボードという Quick BASIC が確保している作業領域を介して、テキストのブロック・コピー、ブロック・ムーブを行うことができます。テキストのブロック・コピーは、**[CTRL]**+**[INS]**と**[SHIFT]**+**[INS]**を用いて次のように行います。



これに対してブロック・ムーブは指定テキストを削除して、別の位置に移動(ムーブ)するので、**[SHIFT]**+**[DEL]**と**[SHIFT]**+**[INS]**を用いて行います。

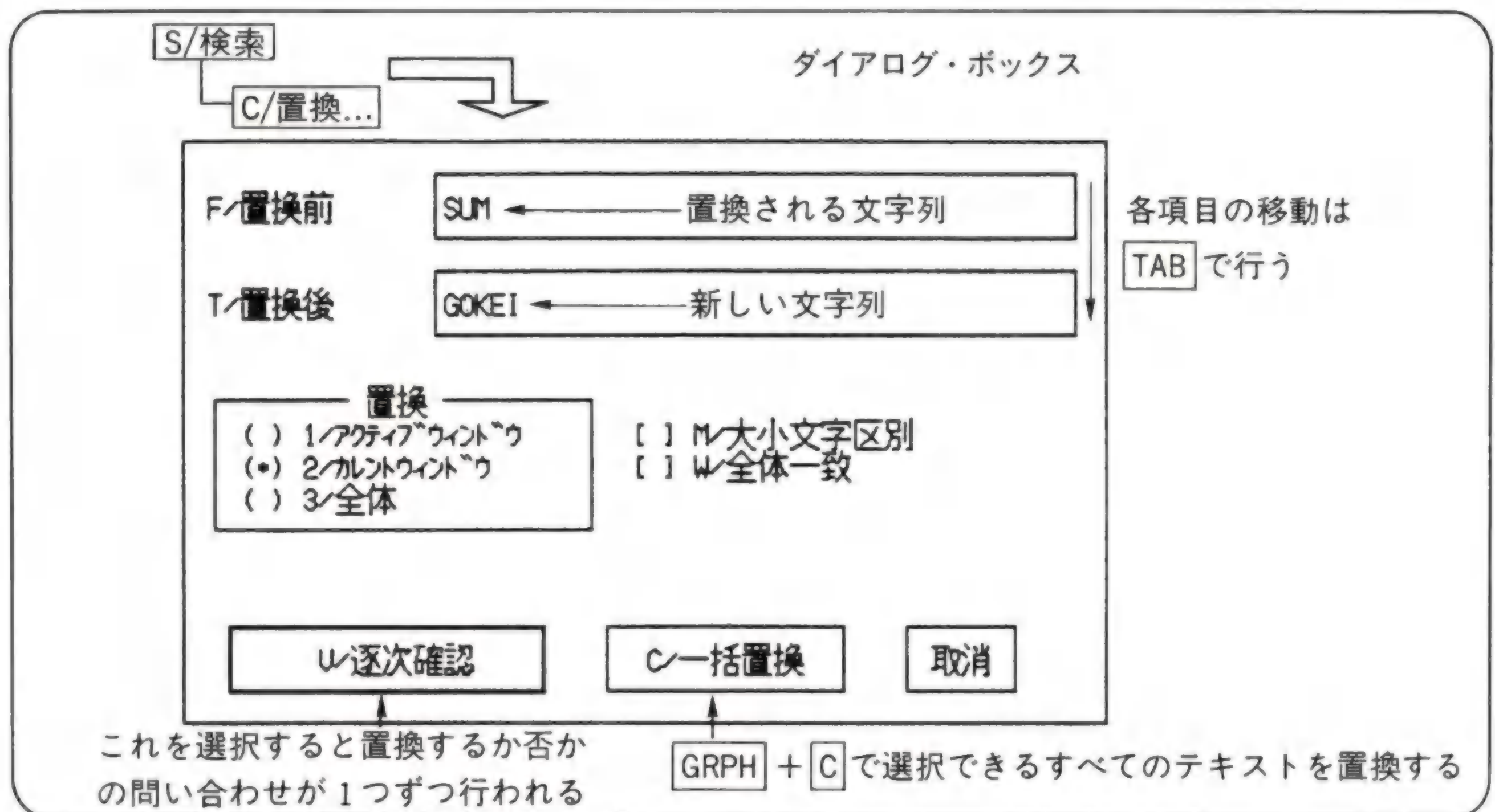
7 文字列のサーチとリプレイス

たとえば、次のようなテキストがあったとします。

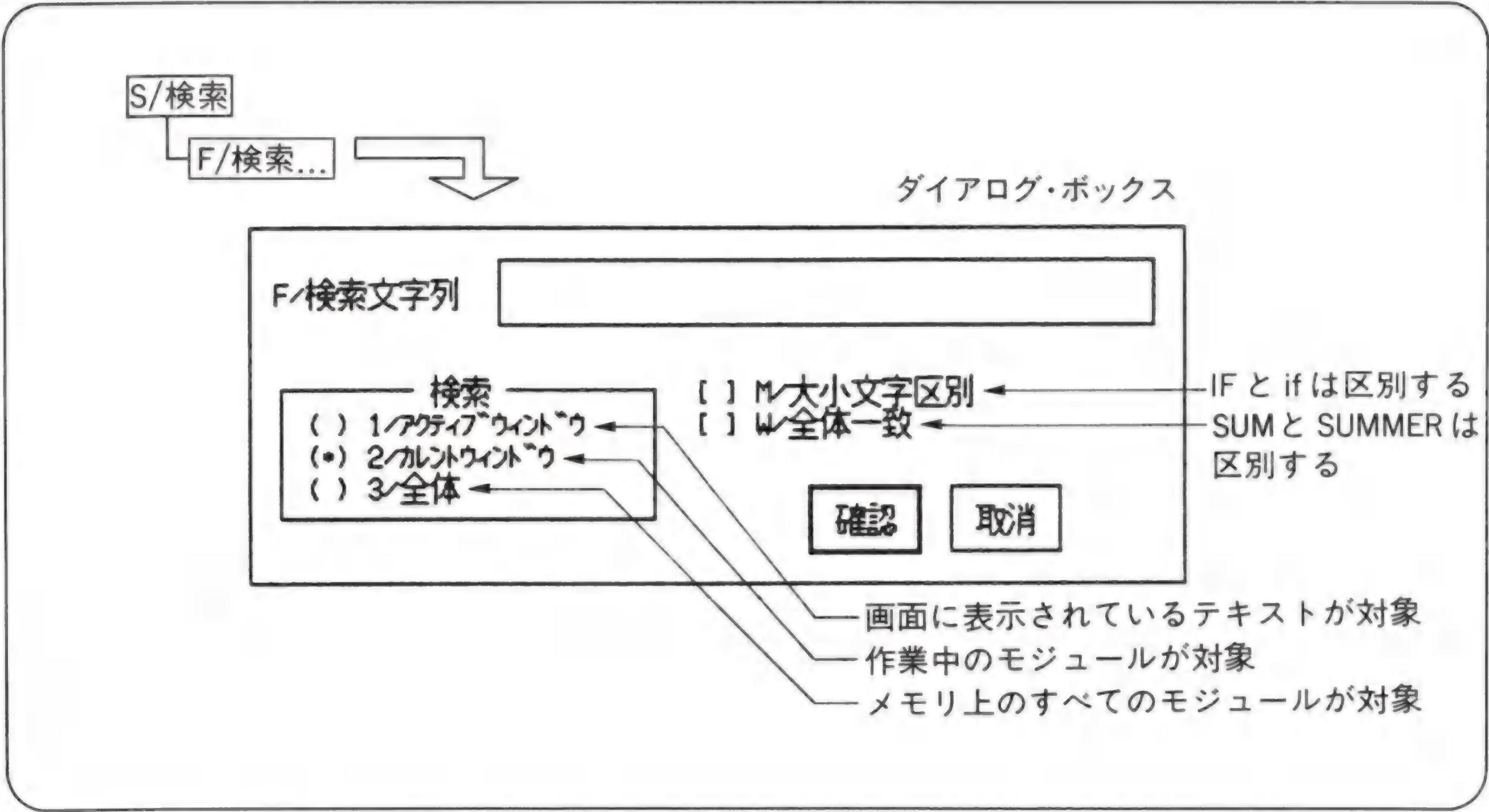
```
SUM = 0
WHILE SUM < 100
  INPUT A
  SUM = SUM + A
WEND
PRINT SUM
```

なんらかの都合で、SUM という文字を GOKEI に変える必要が生じたとき、いちいち SUM を探しだして GOKEI に打ち直してはたいへんですし、見落としも生じるでしょう。このようなときは、文字列のリプレイス・コマンド(**S/検索**—**C/置換...**)を用いて次のように行います。

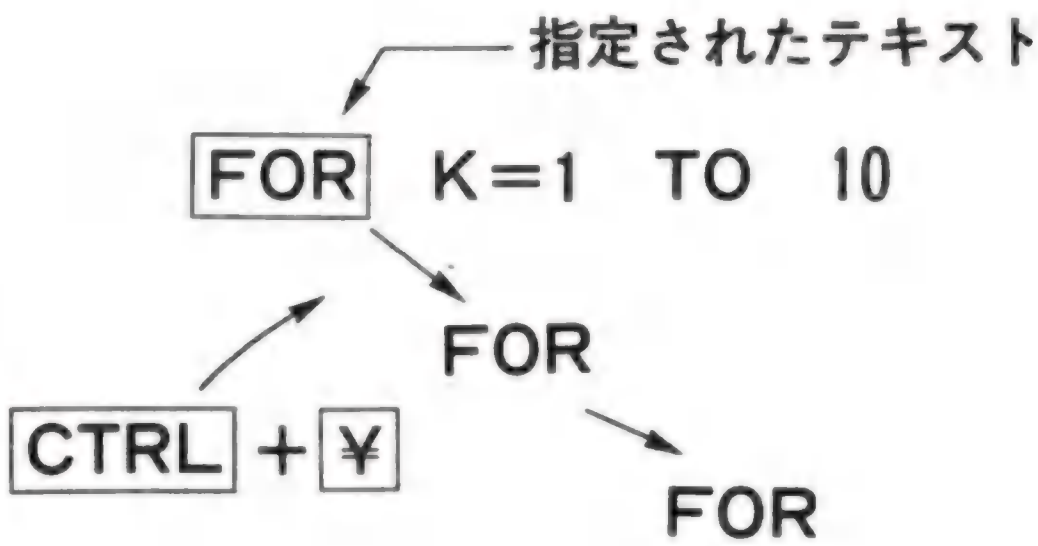
この操作でテキスト中の SUM は一瞬にして GOKEI に置き換えられます。



文字列のサーチは、**S/検索**—**F/検索...**を用いて次のように行います。次の文字列をサーチするには**F・3**を用います。

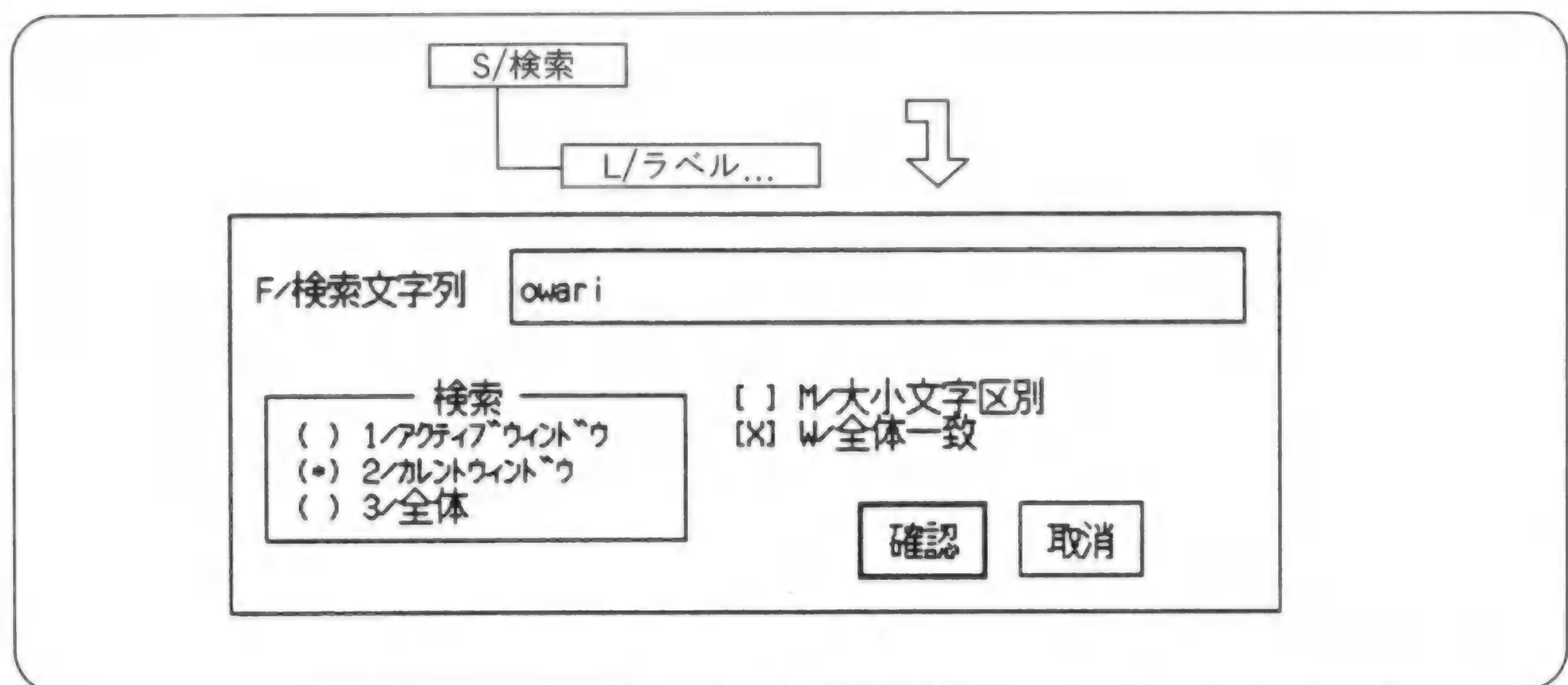


[SHIFT] + 矢印キーにより指定されたテキスト (反転表示されている) をサーチするには、**[S/検索]** → **[S/指定文字列検索]** または **[CTRL] + [¥]** を用います。



8 ラベルの検索

ラベルの検索は[S/検索]—[L/ラベル...]を用います。

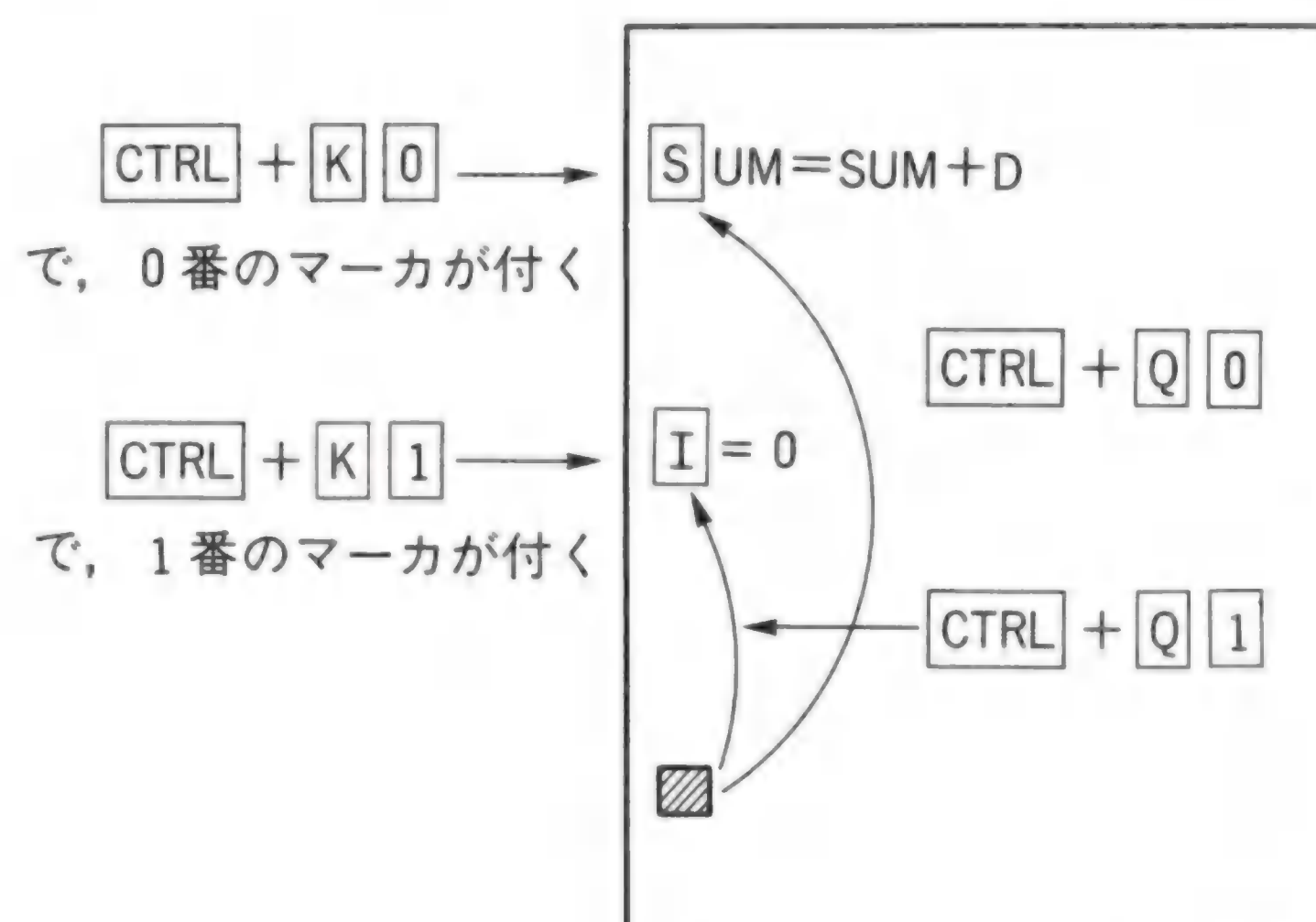


9 その他の機能

■ プレース・マーカ

[CTRL] + [K] に続いて [0] ~ [3] の数字を押すことにより、現在のカーソル行に 0 ~ 3 で認識されるプレース・マーカを設定できます。

逆に [CTRL] + [Q] に続いて [0] ~ [3] の数字を押すことにより、0 ~ 3 で認識されるプレース・マーカが設定されている行にカーソルを移すことができます。



■アン・ドゥ機能

修正したくないテキストを誤って修正してしまったような場合は、カーソルがその行に残っているなら、

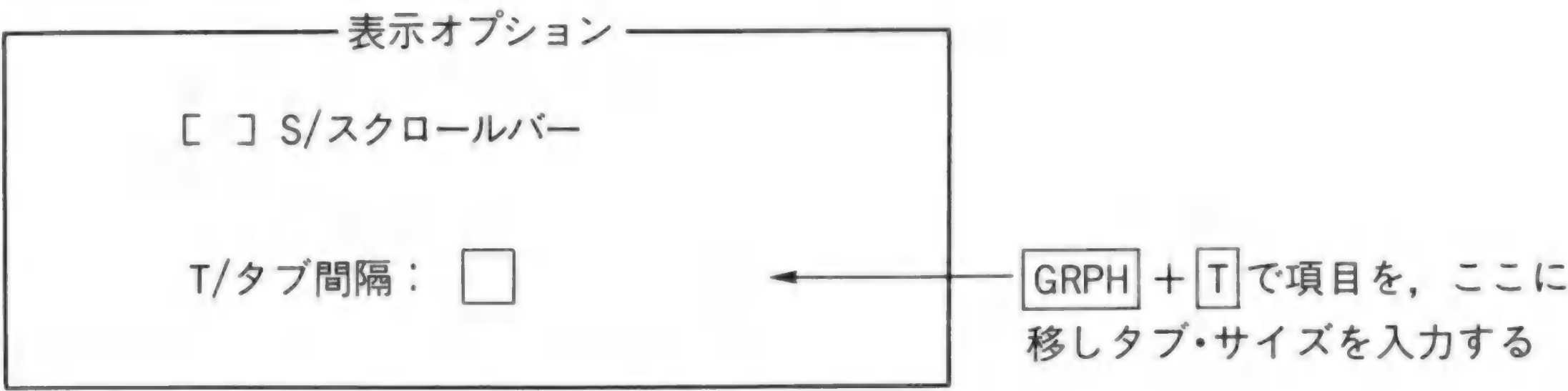
`GRPH` + `BS`

により修正を行う直前の内容に修復することができます。これをアン・ドゥ機能と呼びます。

ただし、`CTRL` + `Y`による1行削除では、カーソルはその行に残らないので`GRPH` + `BS`では修復できません。`SHIFT` + `INS`より修復してください。

■タブ・サイズの変更

Quick BASICでのデフォルトのタブ・サイズは4文字に設定されていますが、`V/表示`—`O/オプション...`メニューにより変更することができます。



注) Ver.4.5では、`O/オプション`—`D/画面表示`で指定します。

■1行につなげる方法

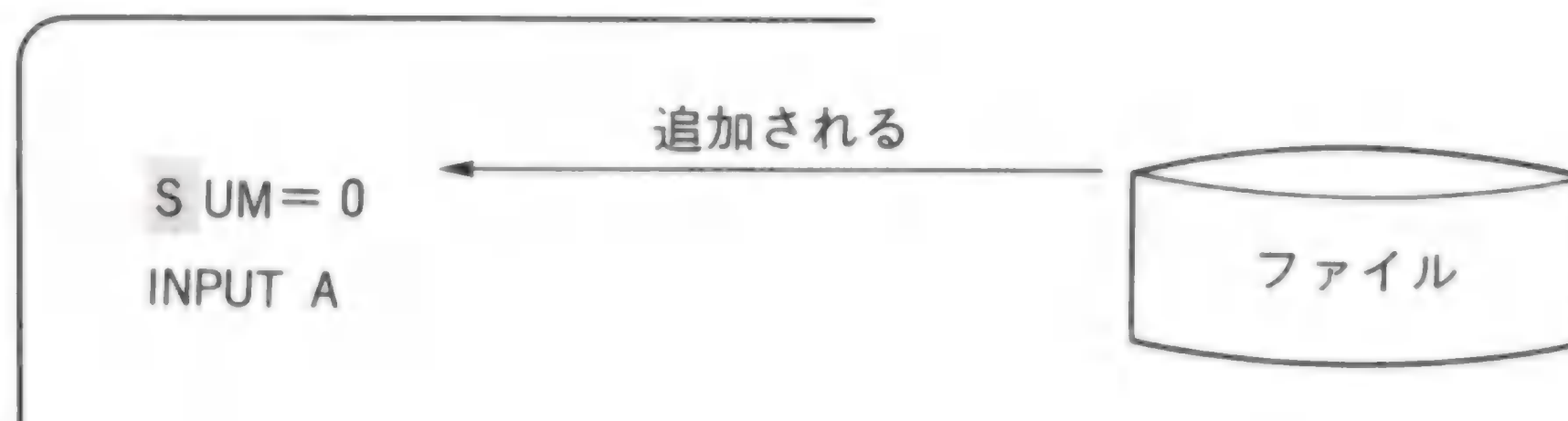
行の途中で`↵`を入力すると、カーソル位置の前後で次のように2行に分かれてしまいます。

```
FOR k = 1 TO 10
    ↓↵
FOR
k = 1 TO 10
```

これを1行に戻すには、カーソルを下の行の先頭に移して`BS`を押すか、カーソルを上の方のテキストの後端に移し`DEL`を押します。

■ほかのファイルをテキスト内に追加する方法

[F/ファイル]—**[M/追加...]**により，ほかのファイルを現在作業中のテキストのカーソル位置の上部に挿入することができます。



■画面表示色のカスタマイズ

[V/表示]—**[O/オプション]**により，画面表示色を変更することができます。
注) Ver.4.5では，**[O/オプション]**—**[D/画面表示]**で指定します。

	表示属性		
	文 字	背 景	
N/テキスト	W/白色	B/黒色	<input type="checkbox"/> H/反転
C/カレントステートメント	G/緑色	B/黒色	<input type="checkbox"/> H/反転
B/ブレークポイント行	R/赤色	B/黒色	<input checked="" type="checkbox"/> H/反転

各ボックスに**[TAB]**キーでカーソルを移し，**[B]**…黒
[W]…白までのイニシャルを入力する

X印があれば反転表示属性がON

■プロシージャのウィンドウを開く

編集画面上で「SUB プロシージャ名()**[↵]**」と入力すると，自動的にプロシージャ・ウィンドウが開きますが，**[E/編集]**—**[S/新規 SUB...]**，**[E/編集]**—**[F/新規 FUNCTION...]**を用いても行えます。

2-6

プログラムの実行

1 インタプリタによるプログラムの実行

編集中のプログラムをインタプリタ (QB) を用いて実行するには、

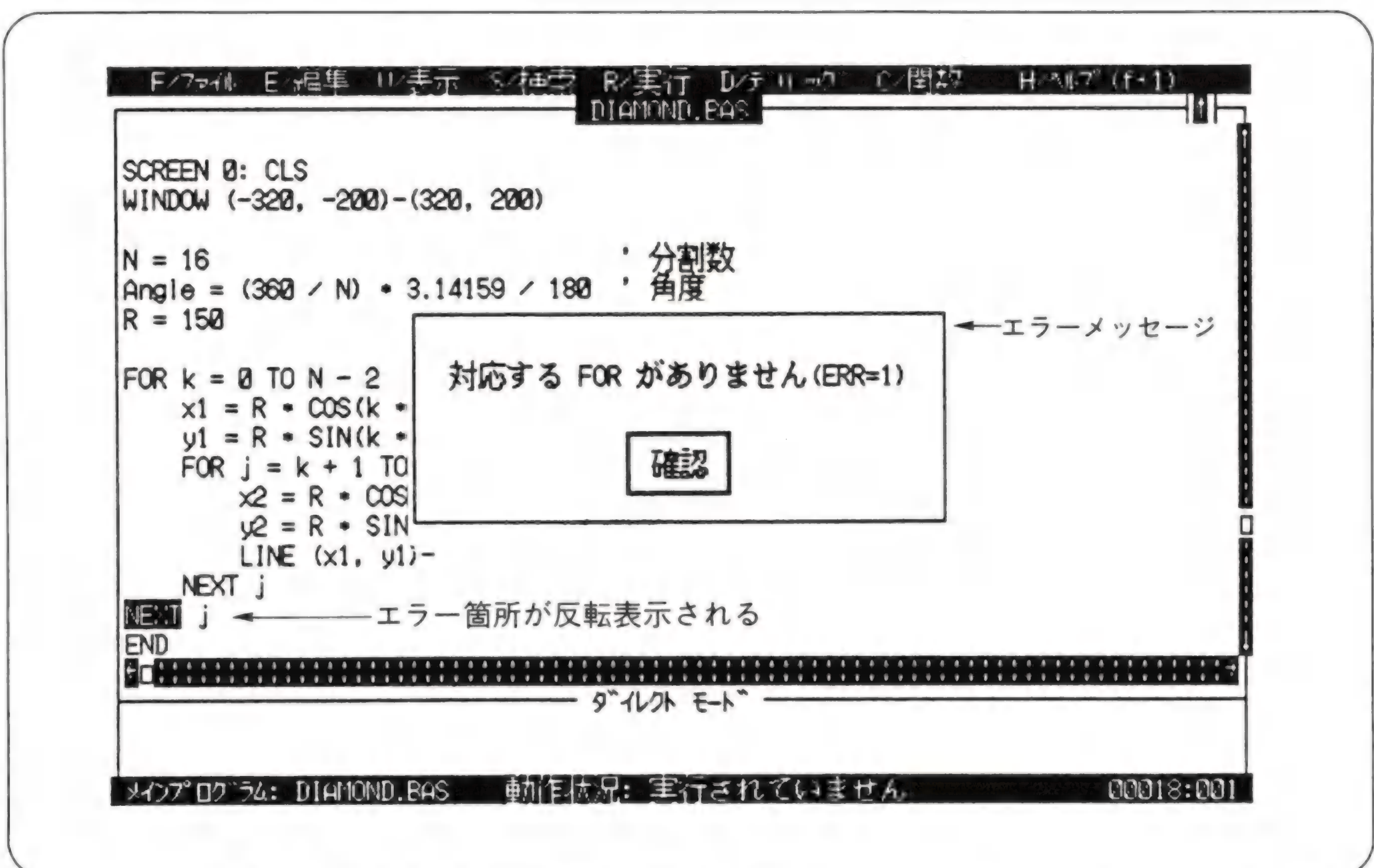
[R/実行] — [S/スタート]

または、

[SHIFT] + [F・5]

で行います。

これでインタプリタが文法チェックを行い、誤りがあれば次のようなエラーメッセージを出します。



[Enter] または [ESC] でエラーメッセージ・ボックスが消え、エラー箇所にカーソルが移りますので、誤りを修正して再実行します。

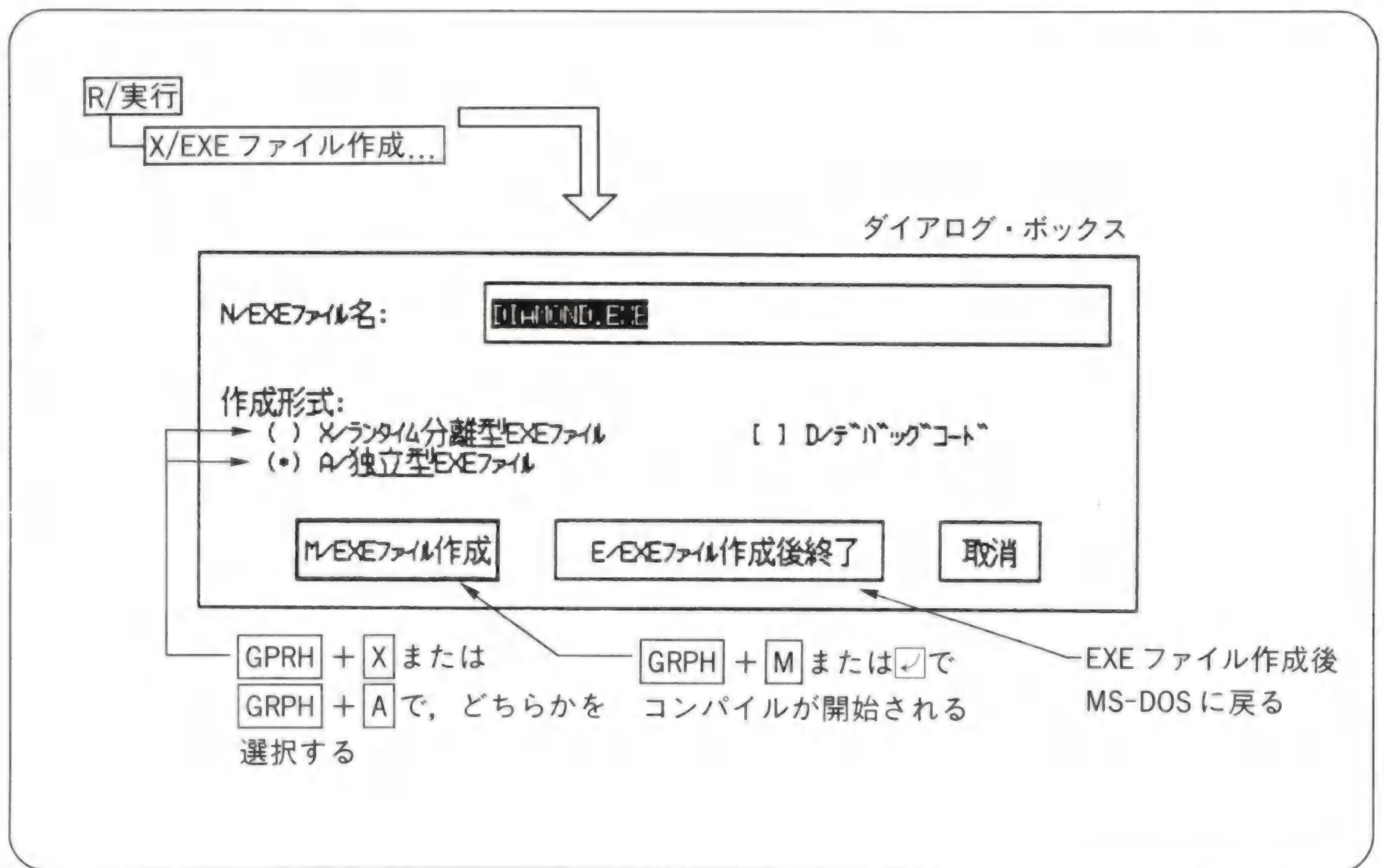
2 実行可能ファイル(.EXE)の作成

インタプリタは、プログラムを開発していく上では対話型でできるため便利ですが、一度完成したプログラムを保存しておく場合にはソースプログラムのレベルでしかできません。したがって、完成したプログラムを実行させる場合には、いちいちインタプリタ(QB)を起動してその上でプログラムを実行しなければならないという、非効率的な面が出てしまいます。

■コンパイラの起動

Quick BASICでは、QB上からBASICコンパイラ(BC.EXE, LINK.EXE)を呼び出し、メモリ上のソースプログラムをコンパイル/リンクして実行可能ファイル(.EXE)を作成することができます。

DIAMOND.BASというプログラムを、QB上からコンパイル/リンクする方法を以下に示します。

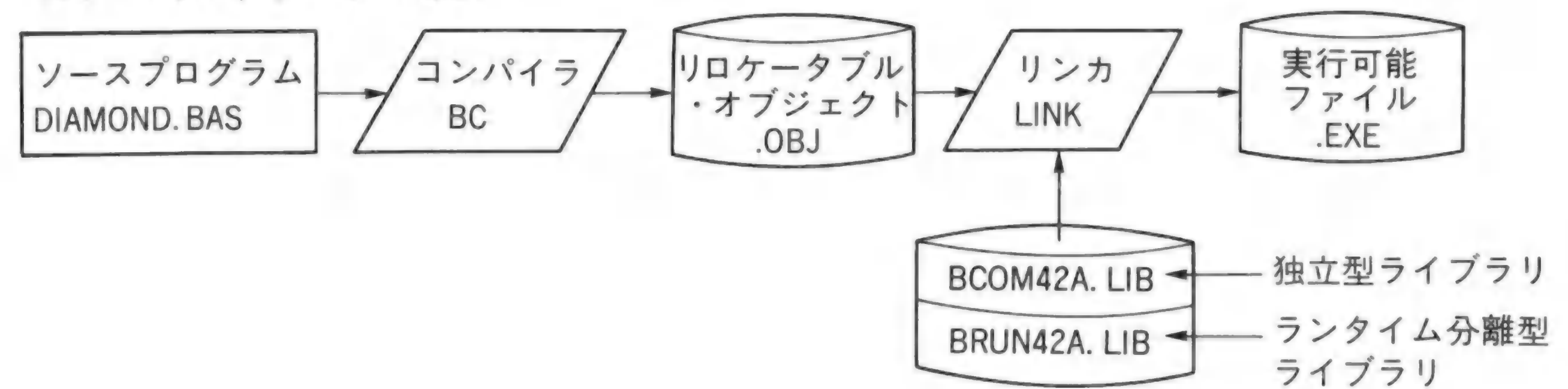


上の操作で次のようにコンパイル/リンクが開始され、ドライブ B に DIAMOND.EXE という実行可能ファイルが作成されます。

・QBからコンパイラ(BC.EXE,LINK.EXE)を起動した様子

```
BC B:¥DIAMOND.BAS/T/C/:512;  
Microsoft (R) QuickBASIC KANJI Compiler Version 4.20  
(C) Copyright Microsoft Corporation 1982-1988.  
All rights reserved.  
  
43527 Bytes Available  
43525 Bytes Free  
  
0 Warning Error(s)  
0 Severe Error(s)  
LINK /EX DIAMOND,A:¥DIAMOND.EXE,NUL,;  
  
Microsoft (R) Overlay Linker Version 3.65  
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.
```

・コンパイル/リンクの流れ



実行可能ファイルは、コマンドラインから、

```
B>DIAMOND ☐
```

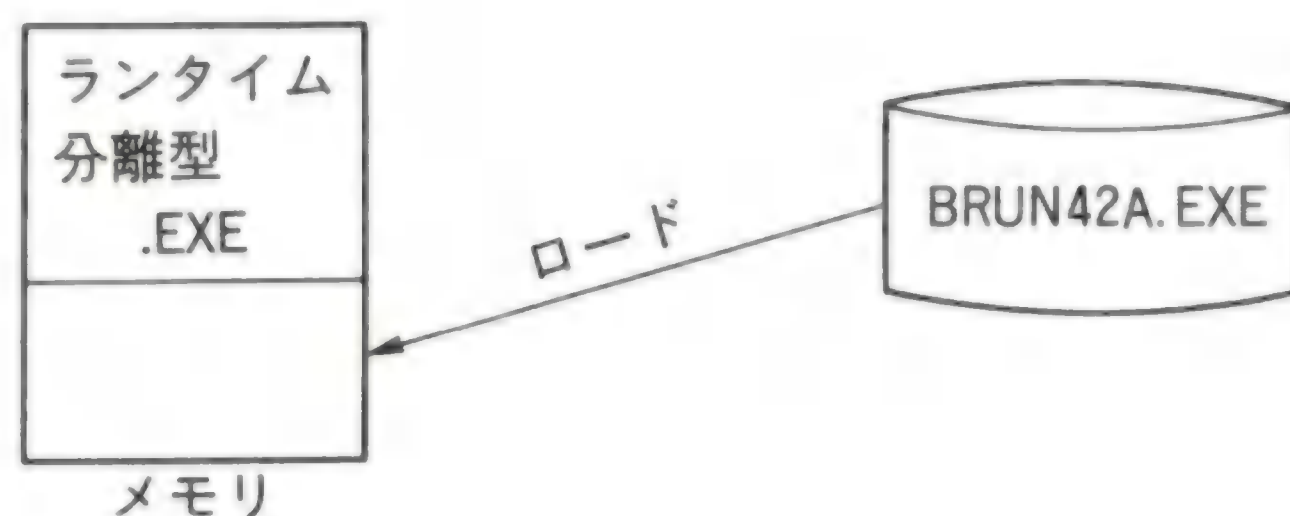
とすることで実行できます。
なお、D/デバッグコード(前頁の[R/実行—X/EXEファイル作成...])を指定しておくと、EXEファイル中に実行時のエラー処理対処のためのコードが埋め込まれます。ただし、このオプションを指定したEXEファイルはCode View デバッガとの互換性はありません。

■実行可能ファイルの作成形式

実行可能ファイルの作成形式は、ランタイム分離型と独立型の2種類が指定できます。使用するライブラリについては、第1章1-2の2の「■ライブラリの選択」(20ページ)を参照してください。

◆ランタイム分離型

リンク時に完全な(実体のある)ライブラリを組み込まずに、プログラム実行時に BRUN42A.EXE というランタイム・ライブラリをメモリ上にロードして、これと協力し合って実行を行うタイプの.EXE ファイルです。



ランタイム分離型の.EXE はファイル・サイズを小さくすることができますが、実行時に必ずランタイム・ライブラリ BRUN42A.EXE がなければなりません。

◆独立型

リンク時に完全なライブラリ (BCOM42A.LIB) を組み込んだ.EXE ファイルです。

独立時の.EXE は、ランタイム分離型の.EXE に比べてファイルサイズは大きくなりますが、単独で(BRUN42A.EXE の協力なしで)実行することができます。

3 クイック・ライブラリの作成

利用価値の高いプログラム(モジュール)を共通の資源として利用できるようにしたものをライブラリといいます。

Quick BASIC では、

- ・ スタンドアロン・ライブラリ(.LIB)
- ・ クイック・ライブラリ(.QLB)

という 2 種類のライブラリを扱うことができます。

QB 統合環境に組み込まれるライブラリをクイック・ライブラリといいます。これに対し、独立して使用されるライブラリをスタンドアロン・ライブラリといいます。スタンドアロン・ライブラリについては、第 3 章 3-3 で説明します。

■ クイック・ライブラリの作成

次のプログラムは、グラフィック画面のハードコピーを行うものです。

```

DECLARE SUB Gcopy ()
DECLARE SUB Lputc (c%)
DECLARE SUB Lputs (a$)

SUB Gcopy      ' グラフィック画面のハードコピー
  Lputs CHR$(&H1B) + "T16"      ' 改行幅
  Lputs CHR$(&H1B) + "D"        ' コピーモード
  FOR x = &H4F TO 0 STEP -1
    Lputs CHR$(&H1B) + "S0400"  ' 8ビットドット対応
    FOR y = 0 TO 399
      DEF SEG = &HA800: a = PEEK(x + y * &H50)      ' Blue
      DEF SEG = &HB000: b = PEEK(x + y * &H50)      ' Red
      DEF SEG = &HB800: c = PEEK(x + y * &H50)      ' Green
      Lputc (a OR b OR c)
    NEXT y
    Lputc &HD: Lputc &HA        ' 改行
  NEXT x
  Lputs CHR$(&H1B) + "A"        ' ノーマル改行幅
  Lputs CHR$(&H1B) + "M"        ' 通常モード
END SUB

SUB Lputc (c%)      ' 直接プリンタ出力
  WHILE (INP(&H42) AND 4) <> 4
  WEND
  OUT &H40, c%
  OUT &H46, 14
  OUT &H46, 15
END SUB

SUB Lputs (a$)      ' 直接文字列出力
  FOR k = 1 TO LEN(a$)
    c% = ASC(MID$(a$, k, 1))
    Lputc c%
  NEXT k
END SUB

```

このプログラムを、クイック・ライブラリ(GCOPY.QLB)として作成するには、**R/実行**—**L/ライブラリ作成...**を用いて次のようにします。

R/実行

L/ライブラリ作成...

Quickライブラリファイル名: gcopy

☐ Delete Back Code

Quickライブラリ作成 Quickライブラリ作成後終了 取消

これで次のように BC(コンパイラ), LINK(リンカ), LIB(ライブラリ・マネジャ)が呼び出され, クィック・ライブラリが作成されます.

```
BC B:¥GCOPY.BAS/T/C/:512;
Microsoft (R) QuickBASIC KANJI Compiler Version 4.20
(C) Copyright Microsoft Corporation 1982-1988.
All rights reserved.

43527 Bytes Available
41633 Bytes Free

0 Warning Error(s)
0 Severe Error(s)
LINK /QU GCOPY,B:¥GCOPY.QLB,NUL,BQLB42.LIB;

Microsoft (R) Overlay Linker Version 3.65
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

LIB B:¥GCOPY.LIB+GCOPY;

Microsoft (R) Library Manager Version 3.08
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.
```

■クィック・ライブラリの利用法

こうして作成したクィック・ライブラリ(GCOPY.QLB)を利用するには, QB 起動時に次のように/L オプションで指定します.

A>QB ☐/LGCOPY ☐

これでライブラリが QB システムに組み込まれますので, ユーザ・プログラムで GCOPY プロシージャを使用するには,

CALL Gcopy

とします. なお, ユーザ・プログラムにおいて,

DECLARE SUB Gcopy()

と宣言しておけば, CALL をつけずに,

Gcopy

だけで, 呼ぶことができます.

なお, QB 起動時の/L オプションにクィック・ライブラリ名を指定しなければ, QB.QLB がデフォルトで選択されます.

4 コマンド・ライン引数の取得

MS-DOS の COPY コマンドで、ファイル a.txt をドライブ B に b.txt としてコピーするには、

```
A>copy a.txt b:b.txt ☐
```

とします。このようにコマンドを投入した行を、コマンド・ラインと呼び、コマンド・ラインからコマンドや実行プログラムに渡されるパラメータを、コマンド・ライン引数と呼びます。

Quick BASIC では COMMAND\$ という関数により、コマンド・ライン引数を取得することができます。

次のプログラムは、

```
A>LIST filename ☐
```

のようにコマンドを与え、filename の内容をタイプするものです。この場合、プログラム中の COMMAND\$ にコマンド・ライン引数の filename が取得されます。

統合開発環境の QB 上でこのコマンド・ライン引数を与えるには、☐R/実行☐C/COMMAND\$入力...☐を用います。

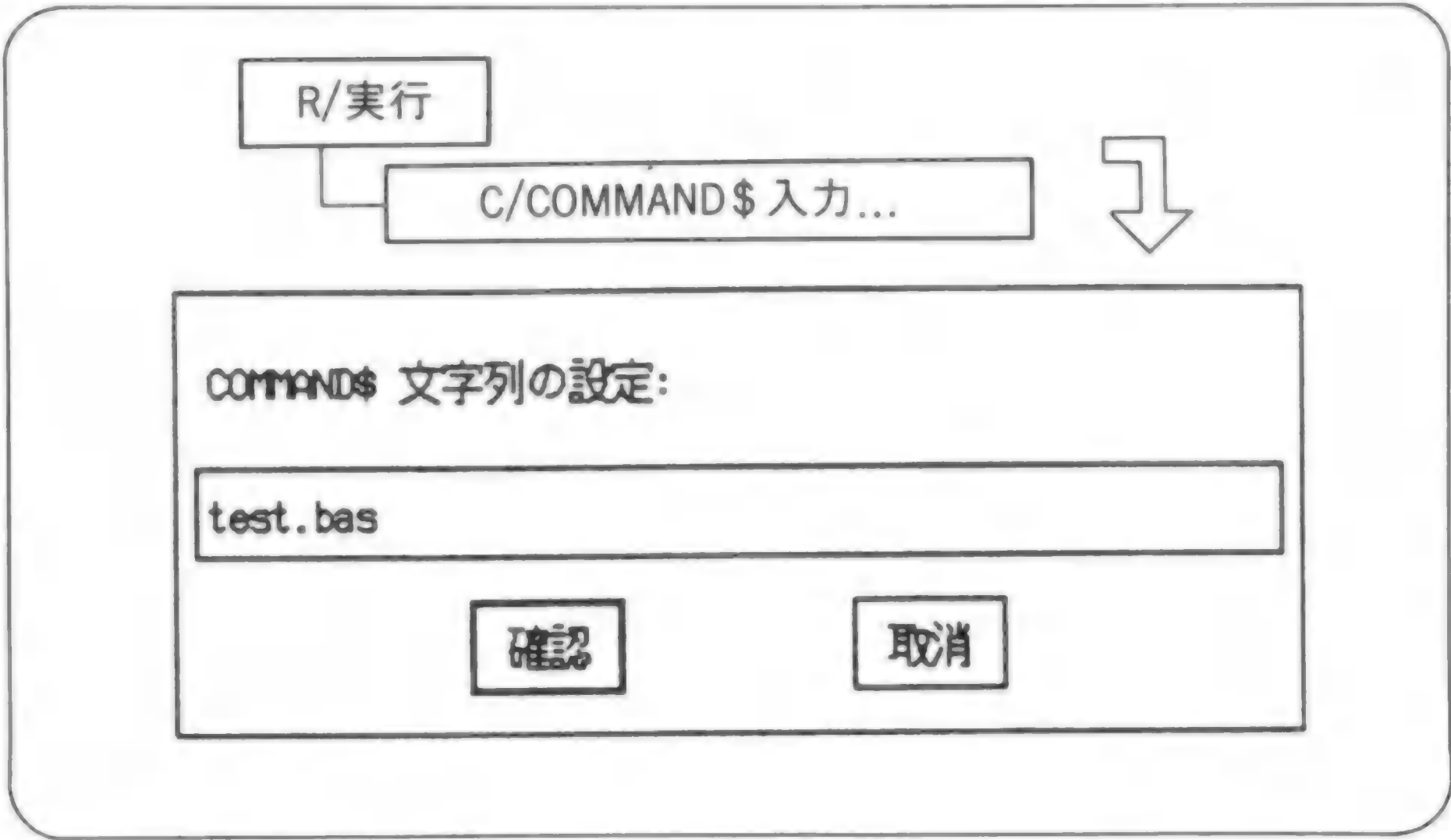
次のプログラムは、コマンド・ラインから与えられるファイル名のファイルの内容を 1 行単位で表示していくものです。

・自作のタイプ・コマンド

```
' タイプ・コマンド(A>LIST filename)

OPEN COMMAND$ FOR INPUT AS #1
WHILE NOT EOF(1)
    LINE INPUT #1, a$
    PRINT a$
WEND
CLOSE #1
```


このプログラムを QB 上で実行するには、**R/実行**—**C/COMMAND\$入力...**を用いて
あらかじめファイル名を設定しておきます。



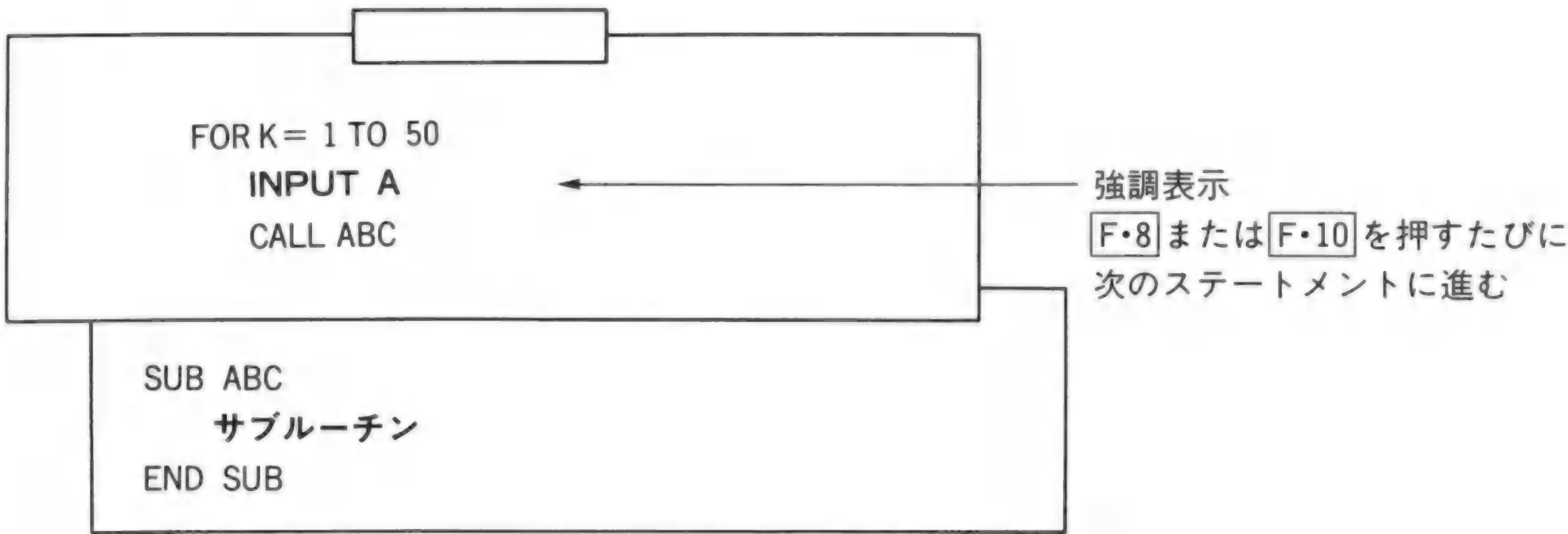
2-7

デバッグ

Quick BASIC には、トレース、ウォッチ、ブレーク・ポイントなどのデバッグ機能があります。

1 トレース機能

トレース機能とは、操作者の指令([F・8]または[F・10])によりプログラムを1ステートメントずつ実行していく機能です。Quick BASIC では、実行(トレース)されているステートメントが強調表示(緑色)されます。



[F・8] と [F・10] の違いは、呼び出したサブルーチン(プロシージャ)の内部もトレースするか否かの点です。[F・8] は呼び出したサブルーチン(上の例ではサブルーチン ABC)の内部もトレースしますが、[F・10] では行いません。

2 プログラムの低速実行

[D/デバッグ]—[T/トレース] を選択してトレース・モードを ON にしておき、[SHIFT]+[F・5] でプログラムを実行すると、1ステートメント単位でゆっくりと行われます。実行中のステートメントは強調表示されますので、プログラムの全体的な流れを見るのに便利です。後述するウォッチ機能と組み合わせて使うと効果的です。

3 ウォッチ機能

プログラム実行中の変数や式の値を、ウォッチ・ウィンドウに刻々と表示する機能をウォッチ機能といいます。

たとえば、変数 x2 の値をウォッチしたい場合は、**D/デバッグ**—**A/ウォッチの追加...**を選択し、次のようにウォッチ式(変数、変数を含む式など)を入力します。

ウォッチウィンドウに登録する式・変数の設定：

x2 ←——ウォッチ式を入力する

変数 y2 についても同様な方法で指定します。

これで次のようにウォッチ・ウィンドウにウォッチ式が登録されます。



このウォッチ式を削除する場合には、**D/デバッグ**—**D/ウォッチの削除...**または**D/デバッグ**—**L/全ウォッチの削除**を用います。

F・8、**F・10**でプログラム・トレースすると、ウォッチ・ウィンドウのウォッチ式の値がプログラムの進行に応じて変化します。

4 ブレーク・ポイント

ブレーク・ポイントとは、プログラムの実行を一時中断する位置(行)のことです。プログラムをデバッグするときに、プログラムの途中経過を見るために止めたい場所があります。こうしたときに、この位置にブレーク・ポイントを設定してからプログラムを実行すると、プログラムの実行がブレーク・ポイントにきたときに中断されます。このとき、ダイレクト・モードで変数の内容を調べるなどのデバッグ作業が行えます。

ブレーク・ポイントの設定は、設定したい位置(行)にカーソルを移動し、

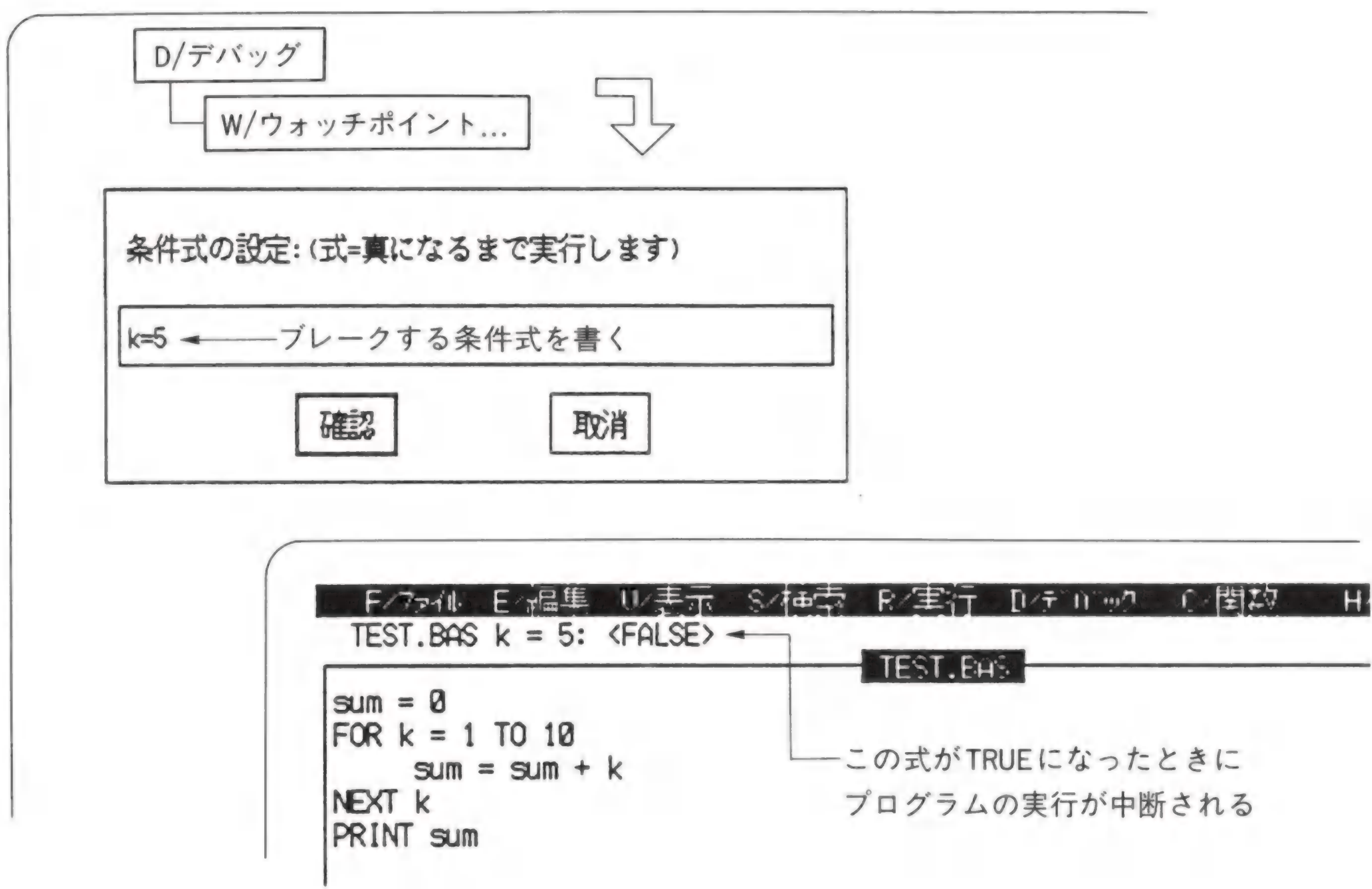
F・9

を押します。これで、この行が赤色のバーで表示されます。すでにブレーク・ポイントが設定されている行で再び**F・9**を押すと、ブレーク・ポイントの解除になります。

ブレーク・ポイントは複数設定することができます。設定したブレーク・ポイントをすべて解除するには、**D/デバッグ**—**C/全ブレーク・ポイントの解放**を用います。

5 ウォッチ・ポイント

ウォッチ・ポイントは条件付きのブレーク・ポイントと考えられます。**D/デバッグ**—**W/ウォッチポイント...**で設定した条件式が真(TRUE)になったときに、プログラムが中断されます。



6 ダイレクトモード

[F・6]でアクティブ・ウィンドウをダイレクトモード・ウィンドウに移すことで、ダイレクトモードでの仕事を行うことができます。

ダイレクトモードとは、BASICのコマンドおよびステートメントを書き、☐を入力することで、そのコマンドおよびステートメントを即座に実行するものです。ダイレクトモードはおもにデバッグ作業で使います。

たとえば、変数 x2, y2の内容を調べたいときは、

ダイレクトモード

? x2,y2 ☐

とします。

ダイレクトモードは1行に収まる範囲で書きます。次のようなマルチステートメント(複数のステートメントを:で区切って1行にしたもの)を書くことができます。

```
for k=0 to 10: ? a(k):next k ☐
```

これで配列 a(0)~a(10)の内容が表示されます。

プログラムを中断し、ダイレクトモードで、プログラム中で使用している変数の値を変更した場合、プログラムを再開したときの変数の値は、変更された値となります。

ダイレクトモードから次のようにプロシージャを呼び出すこともできます。

```
call disp
```

ダイレクトモード・ウィンドウには最大10行まで入力できますが、それぞれの行は互いに独立しています。ダイレクトモード・ウィンドウを拡大するには`CTRL`+`F・10`をくり返して押します。

7 実行の制御

デバッグ中のプログラムは、プログラムの実行、中断、継続がくり返し行われます。これを制御するために次のメニュー(キー)を用います。

R/実行 — R/リスタート

プログラム中のすべての変数をクリアして、プログラムの最初の実行可能ステートメントを実行開始位置に設定します。トレース作業を最初からやり直すときなどに使います。

R/実行 — N/続行 (F・5)

ブレーク・ポイント、STOP 文などにより中断されたプログラムを中断位置から再開します。

F・7

現在プログラムが実行されている行から、カーソルが置かれている行まで実行します。

D/デバッグ — S/カレントステートメント

次に実行される行を、カーソルが置かれている行に設定します。これは、現在の実行行からカーソル行までを実行していくのではなく、その間をスキップします。

V/表示 — N/次のステートメント

プログラムを中断し、デバッグ作業によりカーソルがほかの行に移ってしまったときに、このメニューにより、プログラムの実行を継続する際の実行行にカーソルを移します。

8 ヒストリ

D/デバッグ — H/ヒストリ を ON にすることで、QB は自動的に、実行されたプログラムの最後の20行を記録します。この実行記録をヒストリと呼びます。

このヒストリは、次のキーにより追跡することができます。

SHIFT + F・8

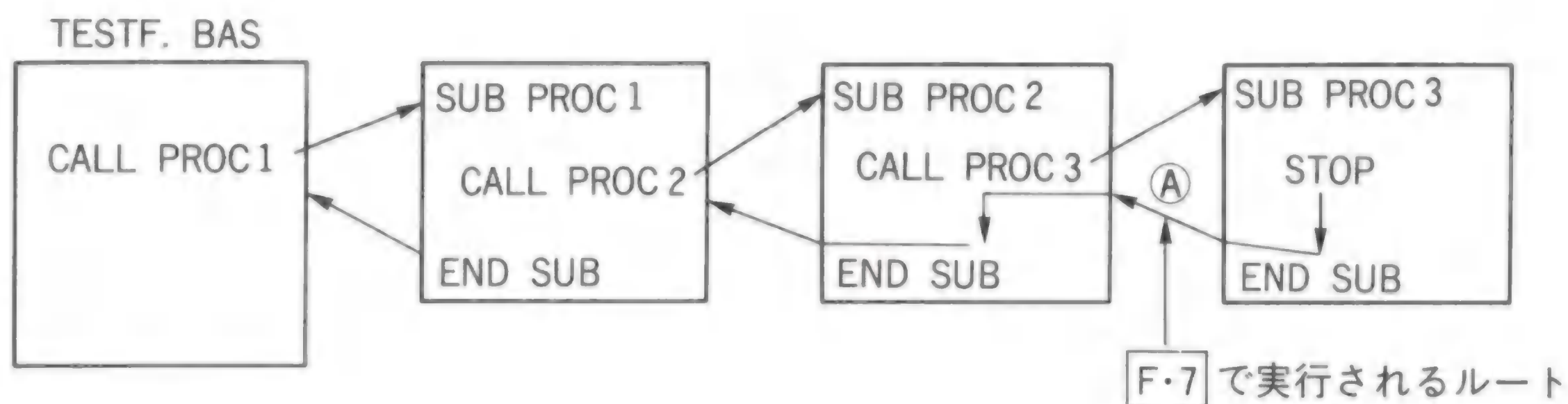
最後に実行された行から、20行前の実行行までを1行ずつカーソルを逆に戻していきます。

SHIFT + F・10

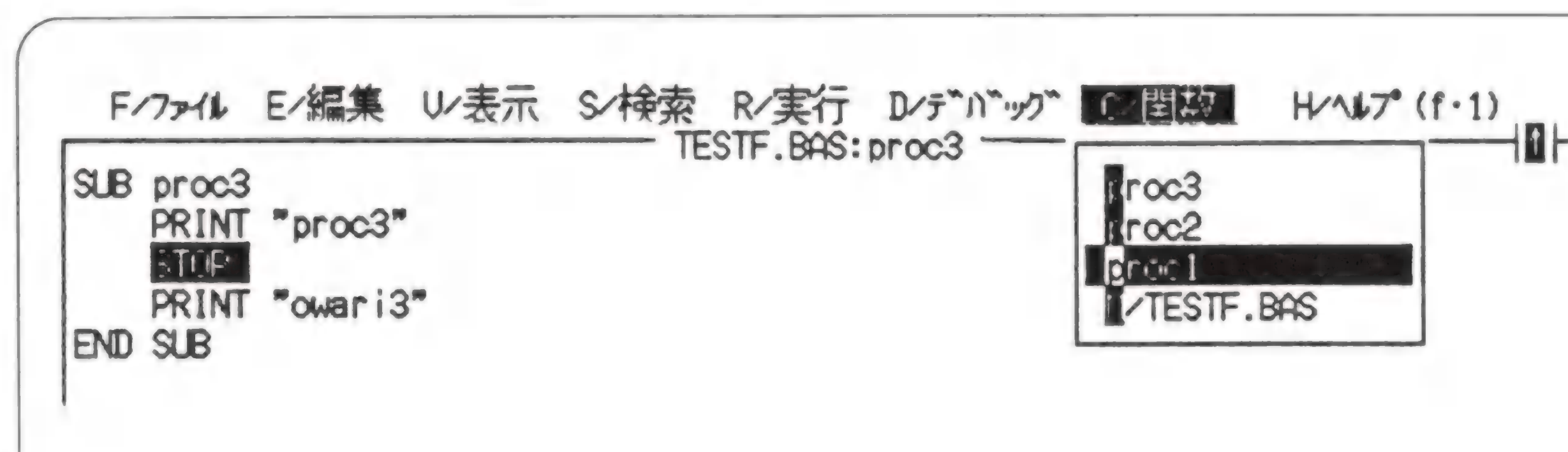
20行前の実行行から最後の実行行に向かってカーソルを1行ずつ移していきます。

9 プロシージャ・コールの追跡

たとえば次のように、プロシージャを次々にネストして呼び出しているようなプログラムがあったとします。



このプログラムを実行すると、PROC3の STOP 文で実行が中断されます。このとき **C/関数** により、次のようなプロシージャのコール順序の一覧が表示されます。一覧表の上になるほど最新に呼び出されたプロシージャです。



一覧表のプロシージャ名にカーソルを移し、**↵**を押すと、そのプロシージャがアクティブ・ウィンドウに表示されます。さらに、**F.7**を押せば、**STOP**などで中断した位置からそのプロシージャの箇所まで、プログラムが実行(プロシージャ・コールをリターンしながら)されます。

たとえば、PROC1にカーソルを移し**↵**を押して、**F.7**を押した場合は、先のプログラムの図の④のルートに沿って、プロシージャ PROC1の CALL PROC 2の次の実行文の直前までを実行します。

2 全般的なヘルプ

[F・1] または [H/ヘルプ] — [G/全般...] により、次のような全般的なヘルプ情報が表示されます。

ヘルプ画面から抜けるには [ESC] を押します。

エディットコマンドとデバッグコマンド

挿入

挿入/上書きモードの切り換え
現在行の上に1行挿入
現在行の下に1行挿入
クリップボードから挿入

INS
HOME CTRL+N
END CTRL+N
SHIFT+INS

選択

文字列/行
単語

SHIFT+ARROW
SHIFT+CTRL+ARROW

削除

現在行 (クリップボードに保管)
ファイルの終端 (クリップボードに保管)
指定文字列 (クリップボードに保管)
指定文字列 (クリップボードに保管しない)

CTRL+Y
CTRL+Q Y
SHIFT+DEL
DEL

エディット

指定文字列

CTRL+INS

検索

指定文字列
再検索

CTRL+F
F・3

デバッグ

出力画面の表示
カーソルのある行まで実行
ブレークポイントの設定/削除
1行ごとの実行
アセンブリごとの実行
前方履歴の実行
後方履歴の実行

F・4
F・7
F・9
F・8
F・10
SHIFT+F・10
SHIFT+F・8

N/次のページ

P/前のページ

K/キートン

取消

注) Ver.4.5では、QB アドバイザーによりもっと詳細なヘルプが得られます。

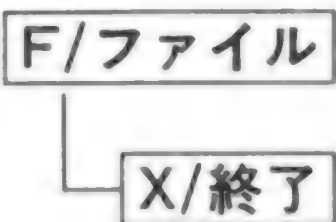
2-9

QB モードから抜ける

QB モードから抜けて MS-DOS に戻るには、次の2つの方法があります。

1 QB を終了して MS-DOS に戻る

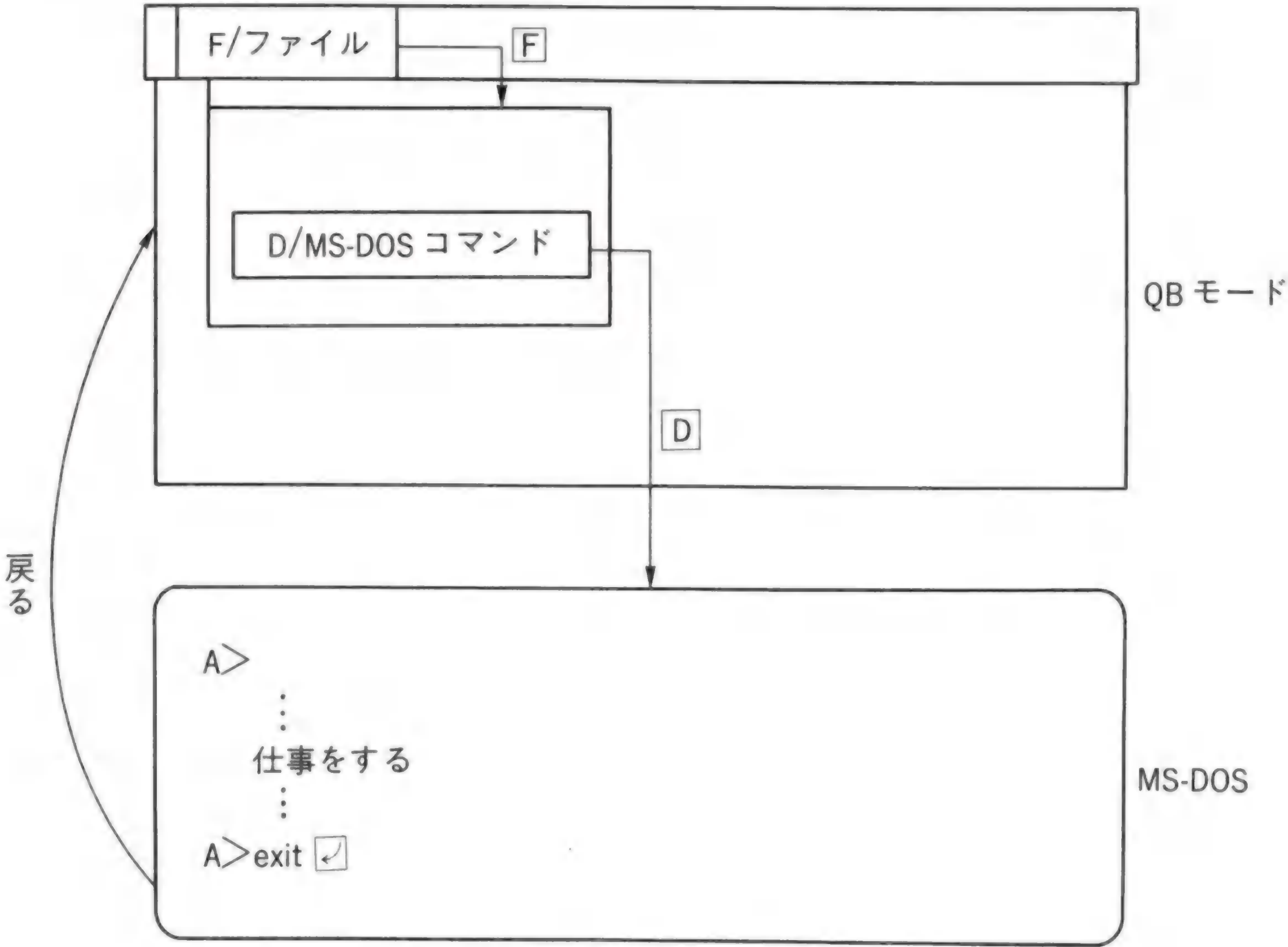
QB モードを終了して MS-DOS に戻るには、次のメニューを選択します。



この場合、再び QB モードに入るには、QB を再起動しなければなりません。

2 QB から MS-DOS に一時的に戻る(OS シェル)

F/ファイル—**D/MS-DOS コマンド**を選択することで、QB モードから一時的に MS-DOS に戻ることができます。



MS-DOS 上でいろいろな仕事をしたあとに、

A>exit

と入力すれば、MS-DOS に移る前の QB モードに瞬時に戻ります。

QB の起動は時間がかかるので、この OS シェル機能はたいへん便利です。

第3章

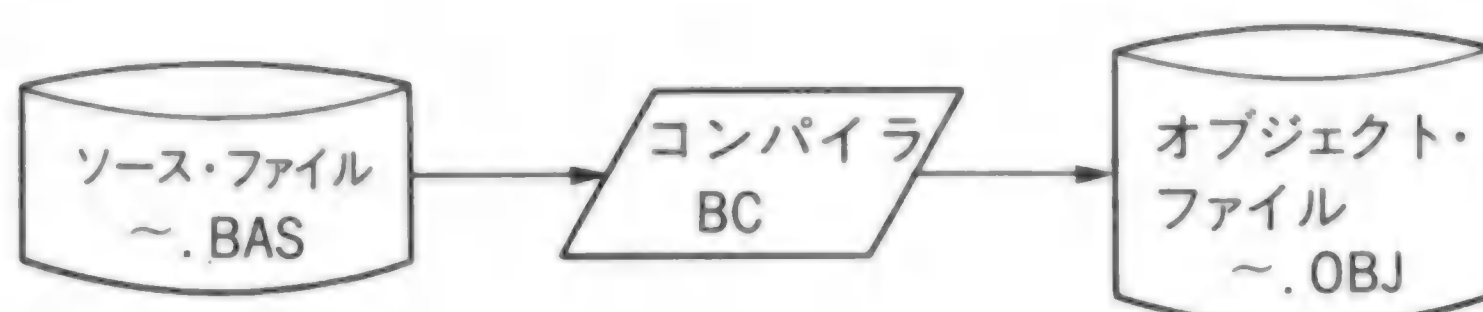
コマンド・
バージョン・
ユーティリティ

3-1

ベーシック・コンパイラ(BC)

BCはBASICのソース・ファイル(.BAS)をコンパイルし、リロケータブル・オブジェクト・ファイル(.OBJ)を作成するものです。

BCにかけるソース・ファイルはテキスト・ファイルでなければなりませんので、QBでプログラムを作る場合は「T/テキストファイル」を指定してセーブしておかなければなりません。



1 BCの起動法

BCの起動法には、次の2つの方法があります。

■方法1 単独で起動する

A>BC☒

Source Filename [.BAS]: ソース・ファイル名☒

Object Filename [filename.OBJ]: オブジェクト・ファイル名☒

Source Listing [NUL.LST]: ソース・リスティング・ファイル名☒

各プロンプトに対応するファイル名を入力します。☒のみ入力すると、[]で囲まれたデフォルトのファイル名が付けられます。ファイル名のタイプを省略すると、[]内のタイプが採用されます。各応答の最後に/で始まるオプションを指定することができます。

プロンプトに対してセミコロン(;)を入力すると、それ以後のプロンプトは表示されず、デフォルトのファイル名が採用されます。

NULとは、ファイルを何も作らないということを意味しています。ファイルはカレント・ドライブのカレント・ディレクトリに作られますので、ほかへ作りたいときはドライブ名またはパス名を付けてください。

ファイル名の大/小文字は区別されません。

例

```

A>bc
Microsoft (R) QuickBASIC KANJI Compiler Version 4.20
(C) Copyright Microsoft Corporation 1982-1988.
All rights reserved.
Source Filename [.BAS]: test
Object Filename [test.OBJ]: test
Source Listing [NUL.LST]:

43511 Bytes Available
43139 Bytes Free

0 Warning Error(s)
0 Severe Error(s)

```

■方法2 ファイル名をコマンド・ラインに与えて起動する

A>BC ソース・ファイル名, [オブジェクト・ファイル名], [リスティング・ファイル名][/オプション]

ファイルは方法1の場合と同様のものです。ファイル名を指定せずにカンマ(,)のみを置くと、そのファイル名はデフォルト解釈されます。またセミコロン(;)を置くと、それ以後のファイル名はすべてデフォルト解釈されます。

例

A>BC TEST; ☐

…ドライブ A の TEST.BAS をコンパイルし、TEST.OBJ を作る。

A>BC B:TEST,B;; ☐

…ドライブ B の TEST.BAS をコンパイルし、ドライブ B に TEST.OBJ を作る。

A>BC TEST,,ABC ☐

…ドライブ A の TEST.BAS をコンパイルし、TEST.OBJ を作り、さらに ABC.LST も作る。

2 BC のコンパイル・オプション

コンパイル・オプションは、スラッシュ(/)またはハイフン(-)で始まる次のような文字で指定します。オプション文字の大/小は区別されません。

■/A (アセンブリコードリスティングの付加)

リスティング・ファイルに、次のようなアセンブリ言語コードを埋め込みます。

```
PAGE 1
15 Aug 87
19:04:46
Offset Data Source LineMicrosoft (R) QuickBASIC KANJI Compiler Version 4.20

0030 0006 sum = 0
0030 0006 FOR k = 1 TO 10
0030 **      100002: mov bx,offset <00000000>
0033 **      call __flds
0038 **      mov bx,offset SUM!
003B **      call __fstsp
0040 **      mov bx,offset <0000803F>
```

■/AH (動的配列に対するメモリの解放)

動的配列に利用可能なすべてのメモリを解放します。このオプションを指定しなければ、1つの配列に対する最大サイズは64KBです。

INPUT N

DIM A(N) …動的配列。/AH を付ければ64KB 以上でもメモリオーバ・エラーにならない。

DIM B(30000) …静的配列。64KB 以上はとれない。

■/C:n (通信ポート・バッファのサイズ)

通信ポートの非同期通信アダプタを使用して、リモートデータを受信するためのバッファサイズをnバイトに設定します。デフォルトは256 バイトです。

■/D (ランタイム・エラー用デバッグ・コードの生成)

ランタイム・エラーを調べるための、デバッグ・コードを埋め込みます。これにより **STOP** を押すと、プログラムの実行をブレークすることができます。

■/E /X (RESUME ステートメントの使用)

/E は RESUME linenumber ステートメントを使った, ON ERROR ステートメントをプログラムで使用している場合に指定します.

/X は RESUME, RESUME NEXT, RESUME 0 ステートメントを使った ON ERROR ステートメントをプログラムで使用している場合に指定します.

第5章の5-11を参照してください.

■/FPI /O (ライブラリの指定)

ライブラリとして以下の4種類のどれを使用するかを指定します.

/FPI はエミュレート・ライブラリの指定, /O はスタンドアロン・ライブラリの指定を行います.

/FPI
↓

		代替数値演算	8087/80287エミュレート
/O →	スタンドアロン型	① BCOM42A.LIB	② BCOM42E.LIB
	ランタイム分離型	③ BRUN42A.LIB	④ BRUN42E.LIB

したがって, /O と /FPI の組み合わせは以下の4種類となります.

- デフォルト
- ③
- /O
- ①
- /FPI
- ④
- /O /FPI
- ②

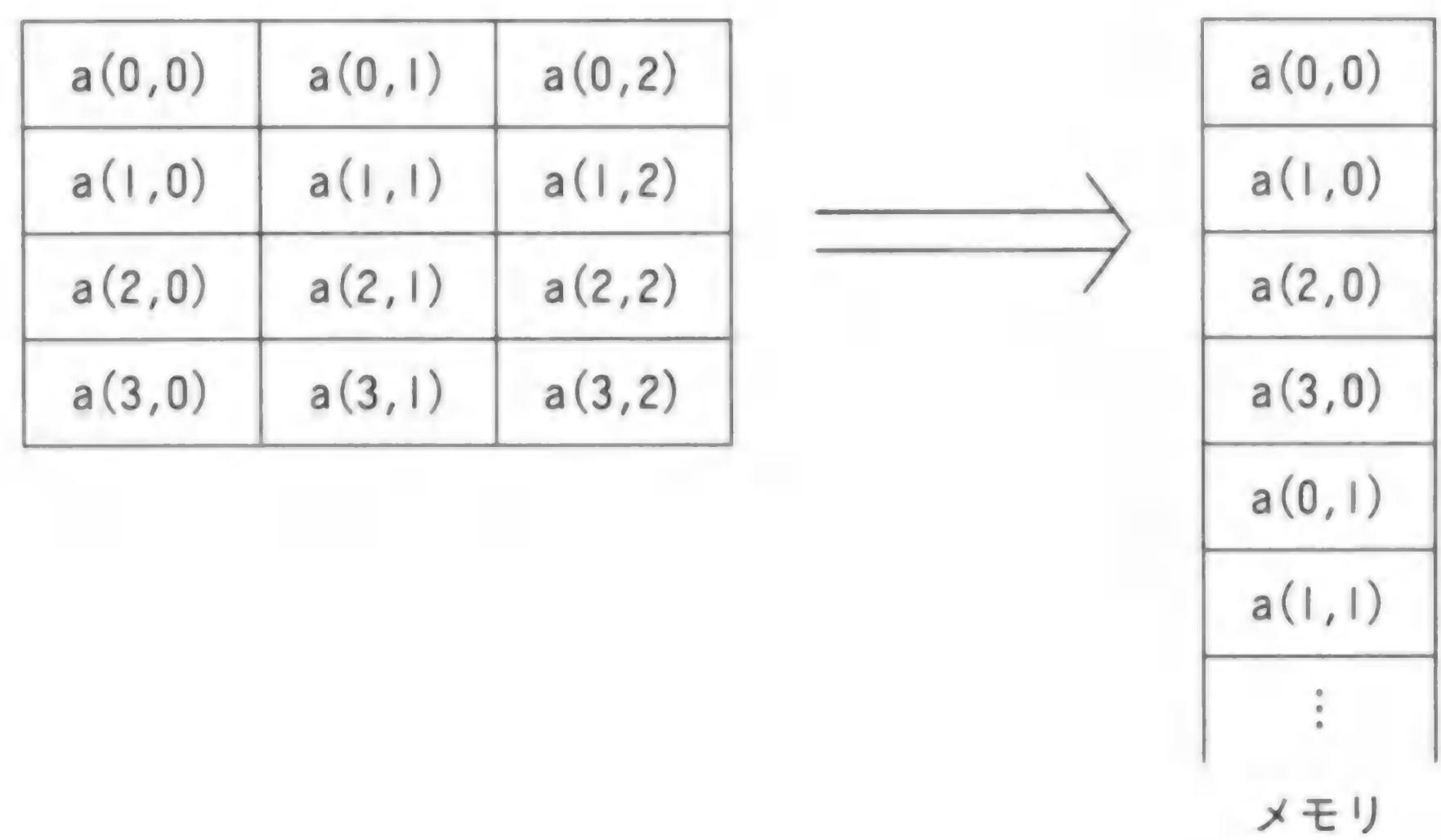
注) Ver.4.5では, 明示的に代替数値演算を指定する /FPA が追加されました.

■/MBF (数値形式の指定)

MK~\$, CV~などの変換関数に対し, IEEE 形式の数値をマイクロソフトバイナリ形式の数値として扱うようにします.

■/R （配列要素のメモリ格納順序）

配列要素のメモリ格納順序を行方向にします。FORTRAN はこの順序で格納します。



■/S （引用文字列の埋め込み方法）

引用文字列をシンボルテーブルでなく、オブジェクト・ファイルに書き込みます。

■/V /W （イベント・トラッピングの使用）

通信(COM), タイマ(TIMER), ファンクションキー(KEY) などのイベント・トラッピングを可能にします。

/V はステートメント間で発生するイベントをチェックし、/W は行間で発生するイベントをチェックします。

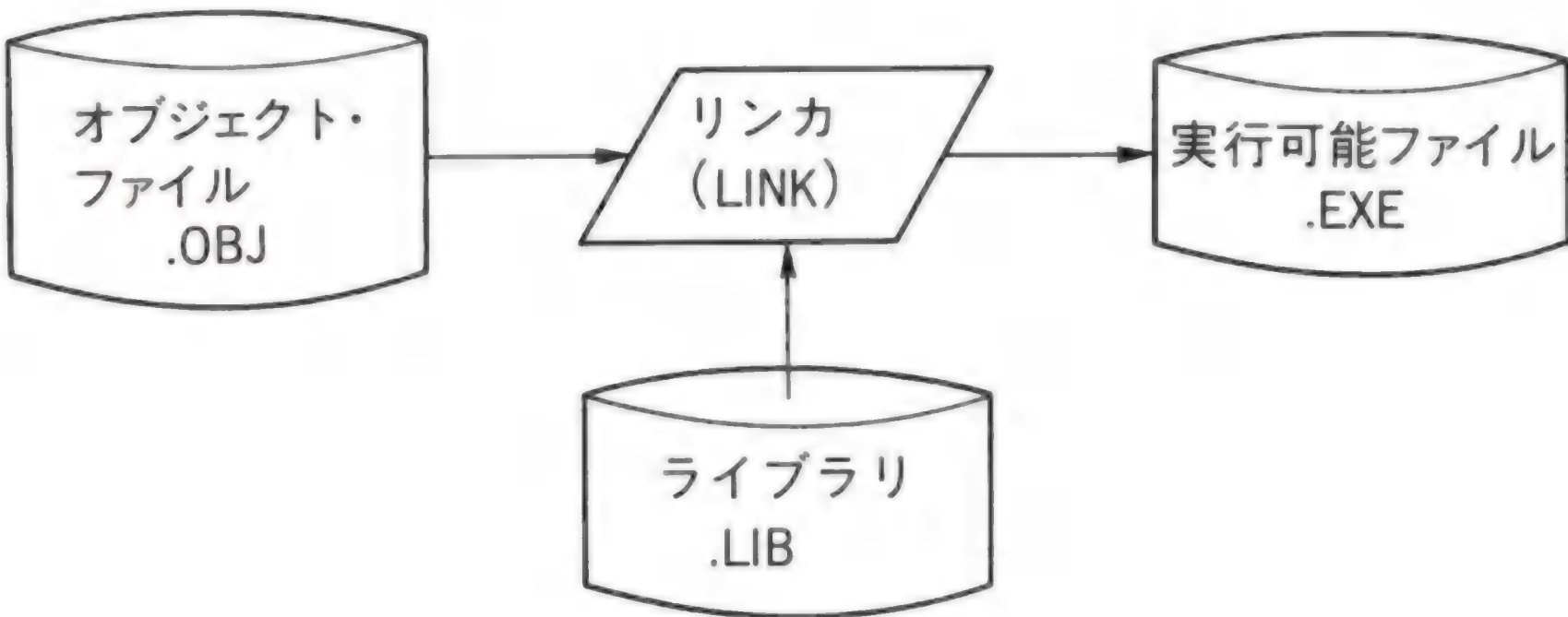
■/ZD /ZI （デバッガ用コードの指定）

/ZD は、SYMDEB でデバッグするためのデバッグ情報を埋め込みます。

/ZI は、Code View でデバッグするためのデバッグ情報を埋め込みます。

3-2 リンカ(LINK)

リンカは、リロケータブル・オブジェクト・ファイル(.OBJ) から実行可能ファイル(.EXE)を生成します。その際必要に応じてライブラリを結合します。



1 LINK の起動法

LINK はデフォルト・ライブラリおよび Libraries プロンプトに対しファイル名のみ与えられたライブラリについては、次の順序でライブラリ・ファイルを探し、ユーザ・オブジェクト・ファイルに結合します。

- 1 カレント・ディレクトリ
- 2 Libraries [.LIB]プロンプトに対して指定したディレクトリ
- 3 LIB 環境変数で指定されているディレクトリ

パスの指定されたライブラリ・ファイルは、上の順序とは関係なく指定されたパス位置を探します。

Quick BASIC の標準ライブラリ(BCOM42A.LIB, BCOM42E.LIB, BRUN42A.LIB, BRUN42E.LIB)は、コンパイル時に、どのライブラリを結合するかの指示がオブジェクト・ファイル中に埋め込まれているので、Libraries プロンプトに対して指定する必要はありません。

LINK に入力する 4 つのファイルの意味は次のとおりです。

▼ LINK に入力するファイル

フ ァ イ ル	機 能
オブジェクト・ファイル Object Modules[.OBJ]	リンクするオブジェクト・モジュール。オブジェクトが複数ある場合は、プラス（+）または空白（ <u> </u> ）で区切る。リンクは並べられた順に同じクラス名のセグメントをまとめてロードする
ラン・ファイル Run File[Object.EXE]	実行可能ファイル。ファイル・タイプには.EXE 以外を与えない方がよい。ファイル名を省略するとオブジェクト・ファイル・リストの先頭のファイル名が採用される
リスト・ファイル List File[NUL. MAP]	ロードされたモジュールの各セグメントに対するエントリをマップしたリスト・ファイル
ライブラリ・ファイル Libraries[.LIB]	ライブラリ・ファイルはライブラリ・マネジャ LIB により作成されたものでなければならない。ライブラリが複数ある場合は、プラス（+）または空白（ <u> </u> ）で区切る

LINK の起動法には次の 3 つの方法があります。

■方法 1 単独で起動する

A>LINK ☒

Object Modules[.OBJ]: オブジェクト・ファイル・リスト名 ☒

Run File[Object. EXE]: ラン・ファイル名 ☒

List File [NUL. MAP]: リスト・ファイル名 ☒

Libraries [.LIB]: ライブラリ・ファイル・リスト名 ☒

各プロンプトに対応するファイル名を入力します。 ☒ のみ入力すると、 [] で囲まれたデフォルトのファイル名が付けられます。ファイルのタイプを省略すると、 [] 内のタイプが採用されます。各応答の最後に“/”で始まるスイッチを指定することができます。

プロンプトに対してセミコロン(;)を入力すると、それ以後のプロンプトは表示されず、デフォルトのファイル名が採用されます。

NUL とはファイルを何も作らないということを意味しています。ファイルはカレント・ドライブのカレント・ディレクトリに作られますので、ほかへ作りたいときはドライブ名またはパス名を付けてください。

<オブジェクト・ファイル・リスト名>と<ライブラリ・ファイル・リスト名>に書く各ファイルは空白またはプラス(+)で区切ります。1 行に書ききれない場合は、行の最後に+を置いて ☒ を入力すると次の行に継続できます。

例

```
A>LINK
Microsoft (R) Overlay Linker Version 3.65
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Object Modules [.OBJ]: test
Run File [TEST.EXE]:
List file [NUL.MAP]: test
Libraries [.LIB]:
```

■方法2 ファイル名をコマンド・ラインに与えて起動する

A>LINK [/オプション]オブジェクト・リスト, [ラン・ファイル], [リスト・ファイル], [ライブラリ・リスト]

ファイルは方法1の場合と同様のものです。ファイル名を指定せずにカンマ(,)のみを置くと、そのファイル名はデフォルト解釈されます。また、セミコロン(;)を置くと、それ以後のファイル名はすべてデフォルト解釈されます。

例

A>LINK B : TEST, B : ., MYLIB

…ドライブ B の“TEST.OBJ”をロードし、MYLIB.LIB とリンクして“TEST.EXE”という実行可能ファイルをドライブ B に作成する。

A>LINK TEST+SMOD1+SMOD2 ;

…“TEST.OBJ”, “SMOD1.OBJ”, “SMOD2.OBJ”をリンクして, “TEST.EXE”を作る。

■方法3 応答ファイル名をコマンド・ラインに与えて起動する

A>LINK @filename

filename で示される応答ファイルには、LINK に応答する 4 種類のファイル名をあらかじめエディタで作成しておきます。filename のファイル・タイプは何であってもかまいません。

たとえば、次のような応答ファイル LINKF をエディタで作成したとします。

▼応答ファイル LINKF

B : TEST

B :

MYLIB

このとき、

A>LINK @LINKF ☒

とすると、“B:TEST.OBJ”が“MYLIB.LIB”とリンクされ、“B:TEST.EXE”ファイルが作成されます。

2 LINK のリンカ・オプション

リンカ・オプションはLINK に対し細かな指示をするためのもので、スラッシュ(/)で始まる英数文字列です。オプション文字の大/小は区別されません。

リンカ・オプションは、フル・スペルと省略形の2通りの指定が可能で、一般に省略形を用います。たとえば、/BATCH は/B と省略して書けます。

以下に Quick BASIC のプログラムをリンクする際によく使う、リンカ・オプションを説明します。

■/B (BATCH: リンカの継続)

リンカが、ライブラリやオブジェクト・ファイルを見つけられなくても、そのままリンカ処理を続けさせます。

■/CO (CODEVIEW: デバッガ用オブジェクトの生成)

Code View デバッガ用のオブジェクトを生成します。

コンパイラ・オプション/ZI を指定してコンパイルしたオブジェクトに対して有効です。

■/E (EXEPACK: 実行可能ファイルのパック)

作成する EXE ファイル中の初期化データの中で、同一の文字が並んでいるものを圧縮し、リロケーション・テーブルを最適化します。これによりファイル・サイズの縮小とロード時間の短い EXE ファイルが生成されます。

■/HE (HELP: ヘルプ表示)

利用できるリンカ・オプションの一覧を表示します。

■/INF (INFOMATION: リンク情報の表示)

リンク情報(リンクのフェーズやリンク中のオブジェクト・ファイル名など)を表示します。

■/LI (LINENUMBERS: 行番号付きのリスティング)

ソース・ファイルに対応した行番号付きのリスティング・ファイルを作成します。

■/M[:n] (MAP: マップ・ファイルの作成)

マップ・ファイルを作成します。nはマップ・リスト中のソートされるシンボルの数で1～65535です。デフォルトは2048です。

■/NOD (NODEFAULTLIBRARYSEARCH: デフォルト・ライブラリの無視)

オブジェクト・ファイルに埋め込まれているデフォルトの標準ライブラリの結合指示を無視します。

このオプションを指定した場合は、Libraries プロンプトに対し、次の順序でライブラリを指示します。

1. ユーザライブラリ
2. Quick BASIC の標準ライブラリ

■/NOE (NOEXTDICTIONARY: 拡張ディクショナリを検索しない)

拡張ディクショナリ(リンクが保守するシンボル位置の内部リスト)を検索しません。このオプションはライブラリ・シンボルが再定義されるときに使います。

■/NOI (NOIGNORECASE: 大/小文字の区別)

大文字と小文字を区別します。つまりグローバル・シンボル upper と UPPER は異なるものと判断します。

■/NOP (NOPACKCODE: セグメントのグループ化をしない)**/PAC[:n] (PACKCODE: セグメントのグループ化をする)**

連続する論理コード・セグメントをグループ化する(/PAC)、しない(/NOP)を指定します。

nは新しいグループ化を始める範囲で、デフォルトは64Kです。

■/PAU (PAUSE:リンクの一時停止)

実行可能ファイルをディスクに書き出す前にメッセージを出して、リンク作業を一時停止します。これにより、実行可能ファイルを別のディスクにセーブすることができます。☞を押すとリンクが再開されます。

■/Q (QUICKLIB:クイック・ライブラリの作成)

オブジェクト(.OBJ)を実行可能ファイル(.EXE)にせずに、クイック・ライブラリ(.QLB)として作成します。

/Q オプションを使用する場合は、クイック・ライブラリ・サポートルーチン BQLB42.LIB をライブラリ・リストに必ず指定してください。

例

```
LINK /Q GCOPY, GCOPY.QLB,,BQLB42.LIB
```

…GCOPY.OBJ からクイック・ライブラリ GCOPY.QLB を作成

■/SE:n (SEGMENTS:セグメント数の指定)

プログラムに許されるセグメント数を n (1 ~ 1024) に制限します。デフォルトは 128 です。

3-3 ライブラリ・マネージャ (LIB)

1 ライブラリ・マネージャとは

利用価値の高いルーチンを共通の資源として使用できるようにすることは、プログラム開発の生産性を向上させます。

リロケートブル・オブジェクト(.OBJ) のレベルで LINK 時にリンクすることも考えられますが、これが何本にも増えてくるとひとかたまりのライブラリ・ファイルにしておいたほうが便利です。リロケートブル・オブジェクト(.OBJ)をライブラリ・ファイル(.LIB)にするのがライブラリ・マネージャ(LIB) です。

Quick BASIC では、

- ・ スタンドアロン・ライブラリ(.LIB)
- ・ クイック・ライブラリ(.QLB)

という 2 種類のライブラリを扱うことができます。

ライブラリ・マネージャ LIB は、スタンドアロン・ライブラリを管理するものです。クイック・ライブラリについては第 2 章 2-6 の 3 を参照してください。

さて、ライブラリ・マネージャについて説明する前に、オブジェクト・ファイルとオブジェクト・モジュールという語についてここで定義しておくことにします。

オブジェクト・ファイルとは、コンパイル(またはアセンブル)により生成されたりロケートブル・オブジェクトです。たとえば、

B:MODUL1.OBJ

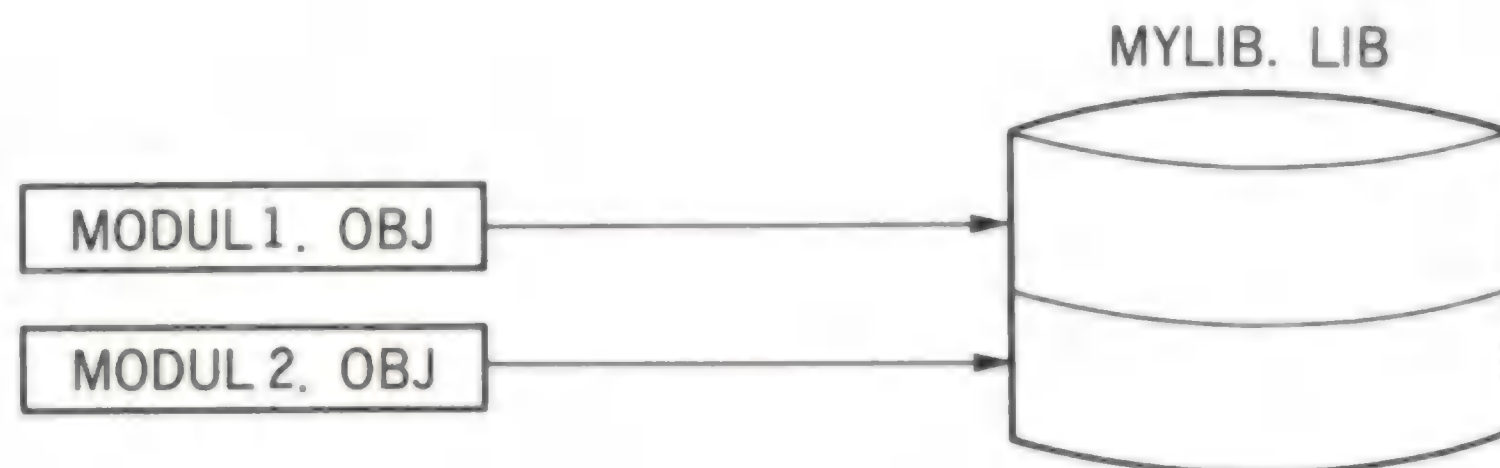
のようにドライブ名とファイル・タイプ(.OBJ)を持つものです。

一方、オブジェクト・モジュールとは、オブジェクト・ファイルを LIB(ライブラリ・マネージャ) によりライブラリとしたときの個々のモジュールを指します。たとえば、上の例のドライブ名とファイル・タイプをとった MODUL1がライブラリ・ファイル内での名前となります。

ライブラリ・マネージャの機能の概要を以下に紹介します。

■ライブラリの新規作成

一般に、ユーザ・ライブラリは何もない状態から新規に作成されていきます。そこで、MODUL1.OBJ と MODUL2.OBJ というリロケートブル・オブジェクトを MYLIB.LIB として登録します。



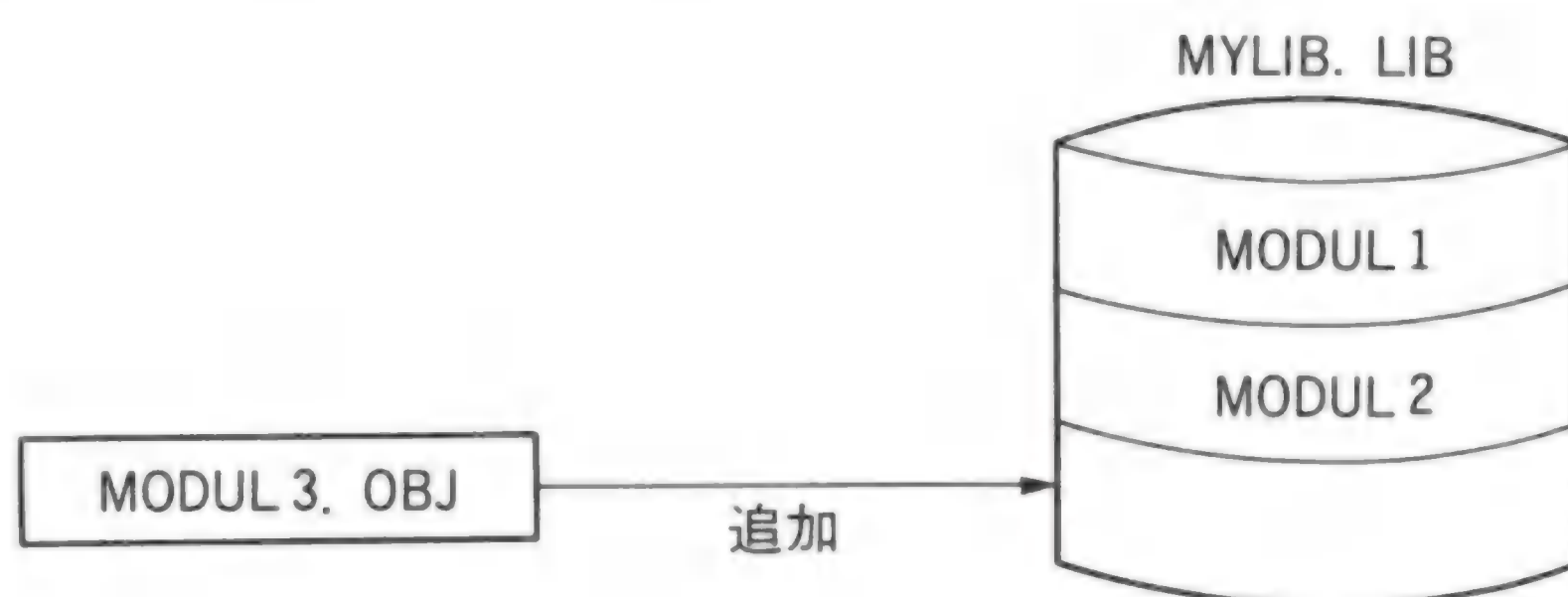
```
A>lib
```

```
Microsoft (R) Library Manager Version 3.11  
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.
```

```
Library name:mylib  
Library does not exist. Create? (y/n) y  
Operations:+modul1+modul2  
List file:
```

■ライブラリへの追加

新規作成されたライブラリ (MYLIB.LIB) へ新しいモジュール (MODUL3.OBJ) を追加します。基本的にはライブラリへの追加も、ライブラリの新規作成も同じ操作です。唯一違うのは、新規作成においてはライブラリ・ファイルが存在しないので、LIB からの問い合わせに対して “Y” と答えることだけです。



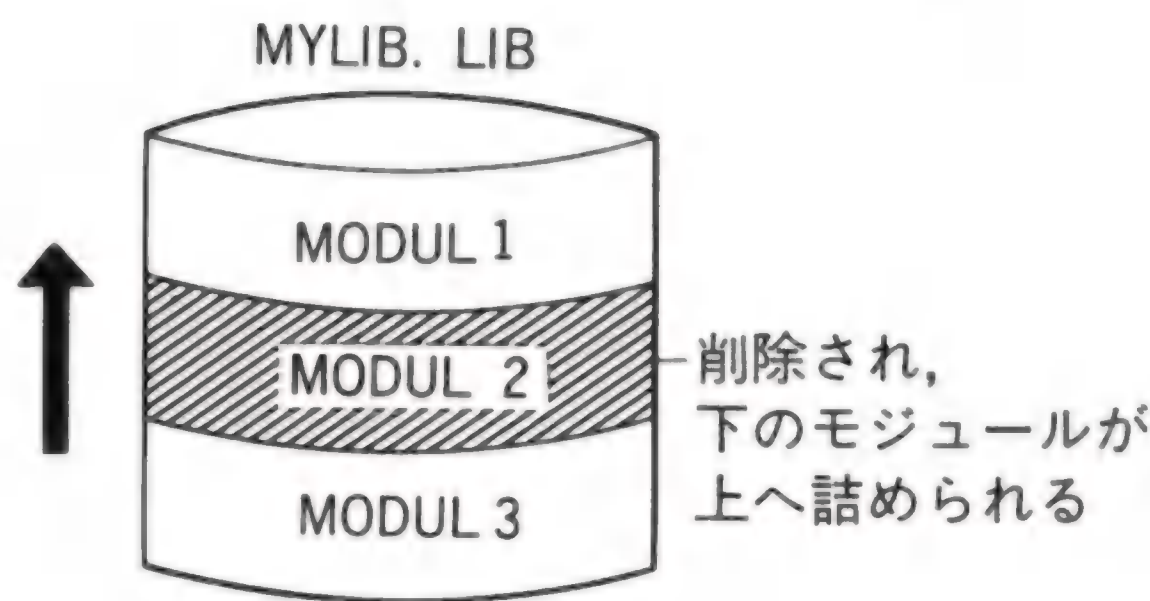
```
A>lib
```

```
Microsoft (R) Library Manager Version 3.11  
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.
```

```
Library name:mylib  
Operations:+modul3  
List file:  
Output library:mylib
```


■ライブラリからの削除

ライブラリ (MYLIB.LIB) から不要になったモジュール (MODUL2) を削除します。MODUL2のあった位置には、下位にあった MODUL3が詰められるので、削除を繰り返してもライブラリ内には空きが生じないようになっています。



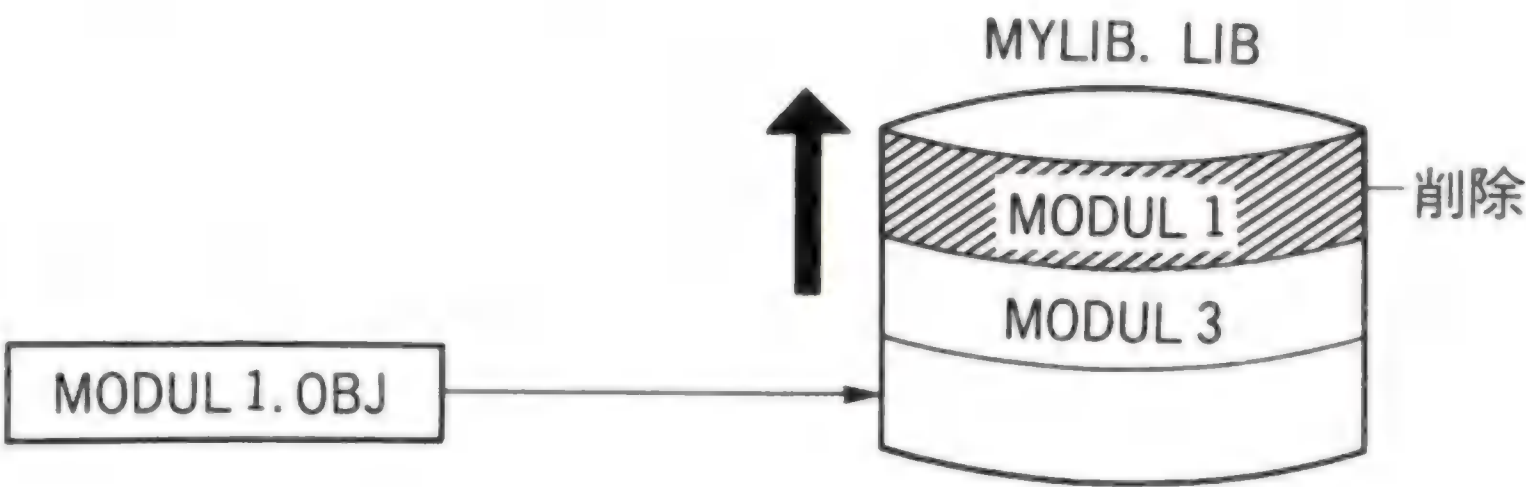
```
A>lib

Microsoft (R) Library Manger  Version 3.11
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Library name:mylib
Operations:-modul2
List file:
Output library:mylib
```

■ライブラリのモジュール置換

ライブラリ MYLIB.LIB の MODUL1を、新しいオブジェクト・ファイル MODUL1.OBJ と置換します。実際には削除と追加が行われます。



```
A>lib

Microsoft (R) Library Manager  Version 3.11
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Library name:mylib
Operations:-+modul1
List file:
Output library:mylib
```


2 LIB の起動法

LIB に入力する 4 つのファイルの意味は以下のとおりです。

▼ LIB に入力するファイル

ファイル	意 味
ライブラリ・ファイル Library name :	ライブラリのファイル名。デフォルトのファイル・タイプは.LIB ライブラリが新規の場合だけこの応答に対し、LIB は問い合わせを出すので“Y”または“y”と答える。それ以外の文字を入力すると処理は中断する
オペレーション Operations :	コマンド・キャラクタとそれに続くモジュール名を並べたもので、LIB に対する処理内容の指示となる。モジュール名のデフォルト・タイプは.OBJ。このプロンプトに <input checked="" type="checkbox"/> のみを入力すると、ライブラリの整合性のチェックのみを行う
リスト・ファイル List file :	ライブラリの中にあるモジュールの PUBLIC シンボルのクロス・リファレンス・リスティングを収めるファイル
出力ライブラリ・ファイル Output library :	現在のライブラリ・ファイル名と異なるライブラリ・ファイル名を指定できる。デフォルト(<input checked="" type="checkbox"/>)は現在のライブラリ・ファイル名でライブラリが作成される。このとき前のライブラリは.BAK というバックアップ・ファイルに変更される

オペレーションに書くコマンド・キャラクタには次のようなものがあります。－＋と－＊は、Ver.3.0における追加キャラクタです。

▼ コマンド・キャラクタ

キャラクタ	機 能
＋	ライブラリの最後部にオブジェクト・ファイルを追加
－	ライブラリの 1 つのモジュールを削除
＊	ライブラリからモジュールを切り出しオブジェクト・ファイルを作成
－＋	ライブラリのモジュールを同名のオブジェクト・ファイルで置き換え
－＊	指定したオブジェクト・モジュールをライブラリから抜き取り、オブジェクト・ファイルに収める
；	残りのプロンプトに対し、デフォルトの応答をする
&	コマンド行に書ききれなくなったときに次の行へ継続

LIB の起動法には次の3つの方法があります。

■方法1 単独で起動する

A>LIB ☐

Library name: ライブラリ・ファイル名[/PAGESIZE:n] ☐

Operations: オペレーション ☐

List file [NUL.MAP]: リスト・ファイル名 ☐

Output library: 出力ライブラリ・ファイル名 ☐

各プロンプトに対応するファイル名またはオペレーションを入力します。☐のみ入力すると,[]で囲まれたデフォルトのファイル名が付けられます。プロンプトに対してセミicolon(;)を入力すると、それ以後のプロンプトは表示されず、デフォルトのファイル名が採用されます。

■方法2 ファイル名をコマンド・ラインに与えて起動する

A>LIB ライブラリ・ファイル名[/PAGESIZE:n][オペレーション]
[, リスト・ファイル][, 出力ライブラリ・ファイル名] ☐

ライブラリ・ファイル名とオペレーションの区切りは、コマンド・キャラクタで分離します。コマンド・ラインの各項目の途中にセミicolon(;)を置くと、それ以後の項目はデフォルト値が採用されます。

例

A>LIB MYLIB -+DISP ; ☐

…ライブラリ MYLIB のモジュール DISP を、新しいオブジェクト・ファイル DISP.OBJ で更新。

A>LIB MYLIB ; ☐

…ライブラリ MYLIB の整合性のチェックのみを行う。

A>LIB MYLIB +B : DISP-MODUL1 ; ☐

…ライブラリ MYLIB にドライブ B の DISP.OBJ を追加し、MODUL1 を削除。

A>LIB MYLIB +MODUL1+MODUL2+MODUL3, MYLIB.LST ☐

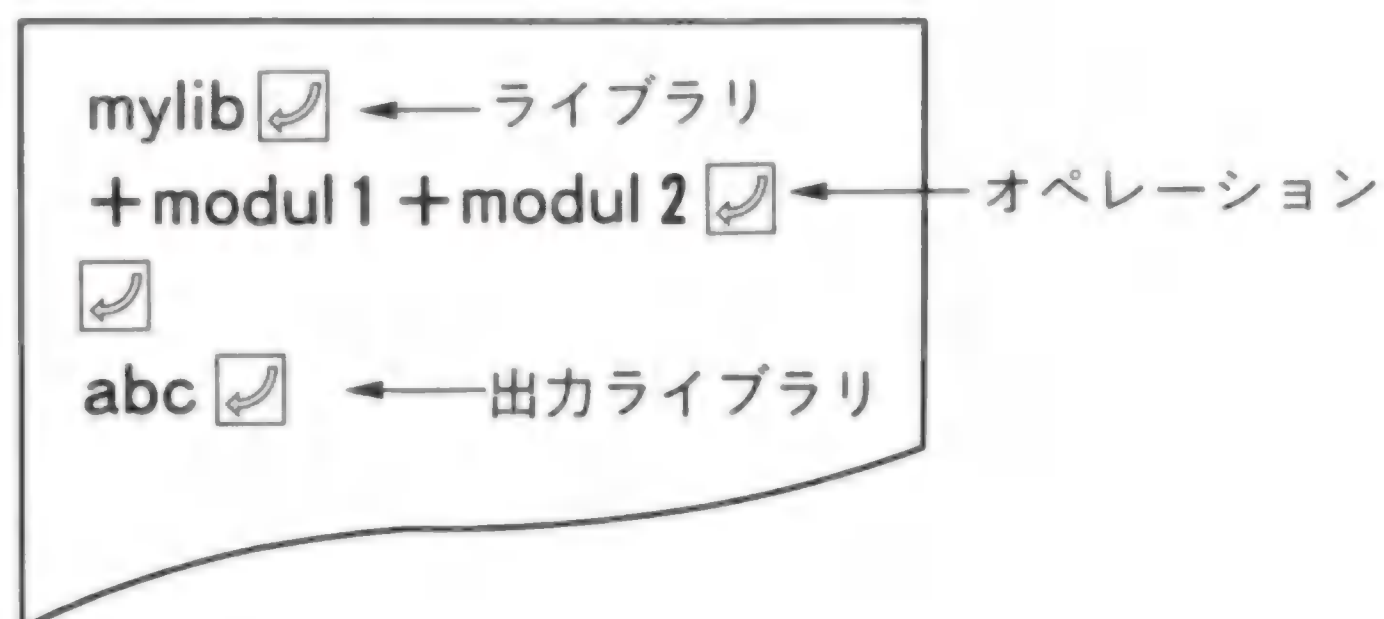
…MODUL1.OBJ, MODUL2.OBJ, MODUL3.OBJ をライブラリ MYLIB.LIB に登録。このとき MYLIB.LST というリスト・ファイルを生成する。

■方法3 応答ファイル名をコマンド・ラインに与えて起動する

A>LIB @filename[↵]

filename で示される応答ファイルには、LIB に応答する 4 種類のファイル名(またはオペレーション)をあらかじめエディタで作成しておきます。filename のファイル・タイプは何であってもかまいません。

filename の例を次に示します。



第4章

Quick BASIC の言語仕様

4-1

プログラムの 基本構成要素

1 Quick BASIC の文字セット

Quick BASIC のプログラムで利用できる文字は、

- ・ アルファベットの大/小文字
- ・ 数字
- ・ 特殊文字

です。特殊文字はプログラム中で特別な意味を持つもので、以下のものです。

特殊文字	名 称	機 能
␣	改行コード	行の終了
!	スペース：空白	文要素の区切り
#	エクスクラメーション：感嘆符	単精度データ型を表す
\$	シャープ：ナンバー	倍精度データ型を表す
%	ドル	文字列データ型を表す
&	パーセント	整数データ型を表す
'	アンパサンド	倍長整数データ型を表す
(シングルのクォーテーション：単一引用符	コメントの開始
)	レフト・パレンシシス：左かっこ	式要素の始め、配列添字の始め
*	ライト・パレンシシス：右かっこ	式要素の終わり、配列添字の終わり
+	アスタリスク	乗算演算子、固定長文字列の長さ指定
,	プラス・サイン：正符号	加算演算子
-	カンマ：句読点	リストの要素の区切り
.	マイナス・サイン：負符号	減算演算子、範囲
/	ピリオド：終止符	小数点、レコードメンバの参照
:	スラッシュ	除算演算子
;	コロンの	ステートメントの区切り
<	セミコロン	リストの要素の区切り
=	レフト・アングル・ブラケット：左山かっこ	比較演算子
>	ライト・アングル・ブラケット：右山かっこ	代入記号、比較演算子
?	イコール：等号	比較演算子
@	ライト・アングル・ブラケット：右山かっこ	PRINT の略字
[クエスション：疑問符	可変長文字列フィールド
]	アット・マーク	配列添字の始め
¥	レフト・ブラケット：左大かっこ	配列添字の終わり
^	ライト・ブラケット：右大かっこ	整数除算演算子
_	円記号	べき乗演算子
	カレット：脱字記号	プログラム行の継続
	アンダー・スコア：下線	

なお、コメント中には、

- ・半角カタカナ
- ・2バイト文字(仮名, 漢字)

を使用することができます。

2 プログラム行の構成

プログラムの各行は次の構成となります。

[行識別子] ステートメント[:ステートメント] ['コメント']

行の終わりは□で示し、1行に最大256文字まで書けます。

1行に複数のステートメント(命令文)を書くことができます。この場合は、各ステートメントをコロン(:)で区切ります。このように、1行に複数のステートメントをコロン(:)で区切って書くことを、マルチ・ステートメントといいます。

●行識別子

QBでは従来型BASICのような行番号は不要です。ただしGOTO文の飛び先として行の先頭に行識別子を置くことができます。

行識別子としては行番号(100, 200など)とラベル(OWARI:など)が使用できます。ラベル名の付け方のきまりは変数名の付け方のきまりと同じです。

●ステートメント(命令文)

ステートメントはプログラムを構成する単一の仕事を記述したもので、命令語とオペランドとからなります。命令語とは、IF, FOR, PRINTなどのBASICがあらかじめ定めている言葉です。オペランドには、定数、変数、関数、そしてこれらを演算子で結んだ式を指定します。

<u>PRINT</u>	<u>A+B</u>
命令語	オペランド
ステートメント(命令文)	

●コメント

コメントは単一引用符(')に続いて書きます。コメントの開始に単一引用符(')を用いた場合はコロン(:)で区切る必要はありませんが、REMを用いた場合はコロン(:)で区切らなければなりません。

Num=40	'生徒の人数
Num=40	: REM 生徒の人数

注)

QBのエディタ以外のエディタを使ってプログラムを作成した場合、1行に収まらない場合に、行末に下線()を置くことで、次の行に続けることができます。このプログラムをQB上にロードすると、下線()は削除され、続く行が結合されて1つの行になります。
ただし、QBのエディタ上で、下線()を使用して行を継続することはできません。

3 予約語

IF, FOR, PRINTなどの命令語と、SIN, COSなどの組み込み関数の関数名は、あらかじめBASICが定めている言葉で、これを予約語と呼びます。
Quick BASICの予約語は以下のとおりです。

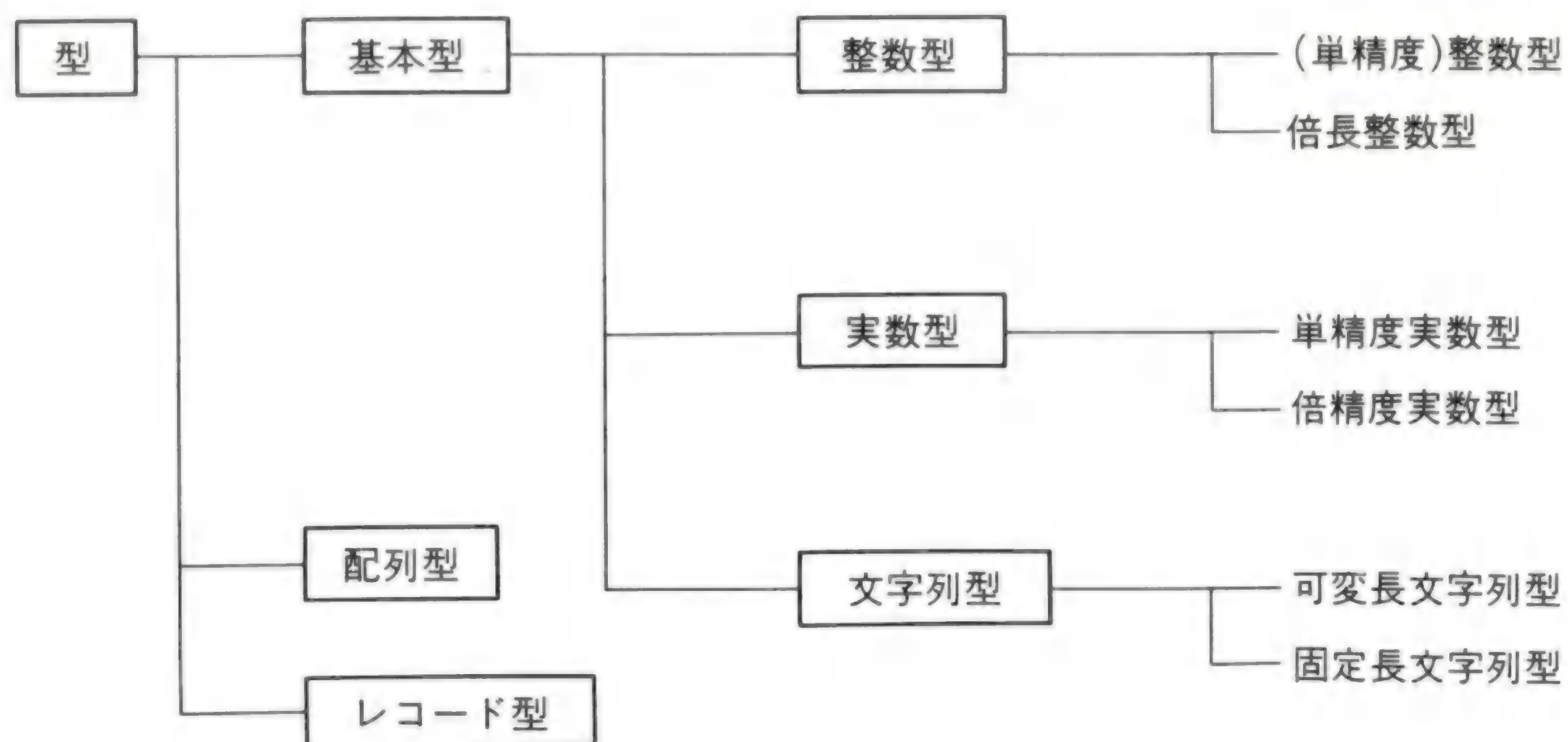
ABS	CSNG	EXIT	LOC
ACCESS	CSRLIN	EXP	LOCAL
ALIAS	CVD	FIELD	LOCATE
AND	CVDMBF	FILEATTR	LOCK
ANY	CVI	FILES	LOF
APPEND	CVL	FIX	LOG
AS	CVS	FOR	LONG
ASC	CVSMBF	FRE	LOOP
ATN	DATA	FREEFILE	LPOS
BASE	DATE\$	FUNCTION	LPRINT
BEEP	DECLARE	GET	LSET
BINARY	DEF	GOSUB	LTRIM\$
BLOAD	DEFDBL	GOTO	MID\$
BSAVE	DEFINT	HEX\$	MKD\$
BYVAL	DEFLNG	IF	MKDIR
CALL	DEFSNG	IMP	MKDMBF\$
CALLS	DEFSTR	INKEY\$	MKI\$
CASE	DIM	INP	MKL\$
CDBL	DO	INPUT	MKS\$
CDECL	DOUBLE	INPUT\$	MKSMBF\$
CHAIN	DRAW	INSTR	MOD
CHDIR	ELSE	INT	NAME
CHR\$	ELSEIF	INTEGER	NEXT
CINT	END	IOCTL	NOT
CIRCLE	ENDIF	IOCTL\$	OCT\$
CLEAR	ENVIRON	IS	OFF
CLNG	ENVIRON\$	KEY	ON
CLOSE	EOF	KILL	OPEN
CLS	EQV	LBOUND	OPTION
COLOR	ERASE	LCASE\$	OR
COM	ERDEV	LEFT\$	OUT
COMMAND\$	ERDEV\$	LEN	OUTPUT
COMMON	ERL	LET	PAINT
CONST	ERR	LINE	PALETTE
COS	ERROR	LIST	PCOPY

PEEK	RIGHT\$	SPACE\$	TROFF
PEN	RMDIR	SPC	TRON
PLAY	RND	SQR	TYPE
PMAP	RSET	STATIC	UBOUND
POINT	RTRIM\$	STEP	UCASE\$
POKE	RUN	STICK	UNLOCK
POS	SADD	STOP	UNTIL
PRESET	SCREEN	STR\$	USING
PRINT	SEEK	STRIG	VAL
PSET	SEG	STRING	VARPTR
PUT	SELECT	STRING\$	VARPTR\$
RANDOM	SETMEM	SUB	VARSEG
RANDOMIZE	SGN	SWAP	VIEW
READ	SHARED	SYSTEM	WAIT
REDIM	SHELL	TAB	WEND
REM	SIGNAL	TAN	WHILE
RESET	SIN	THEN	WIDTH
RESTORE	SINGLE	TIME\$	WINDOW
RESUME	SLEEP	TIMER	WRITE
RETURN	SOUND	TO	XOR

4-2 データ型

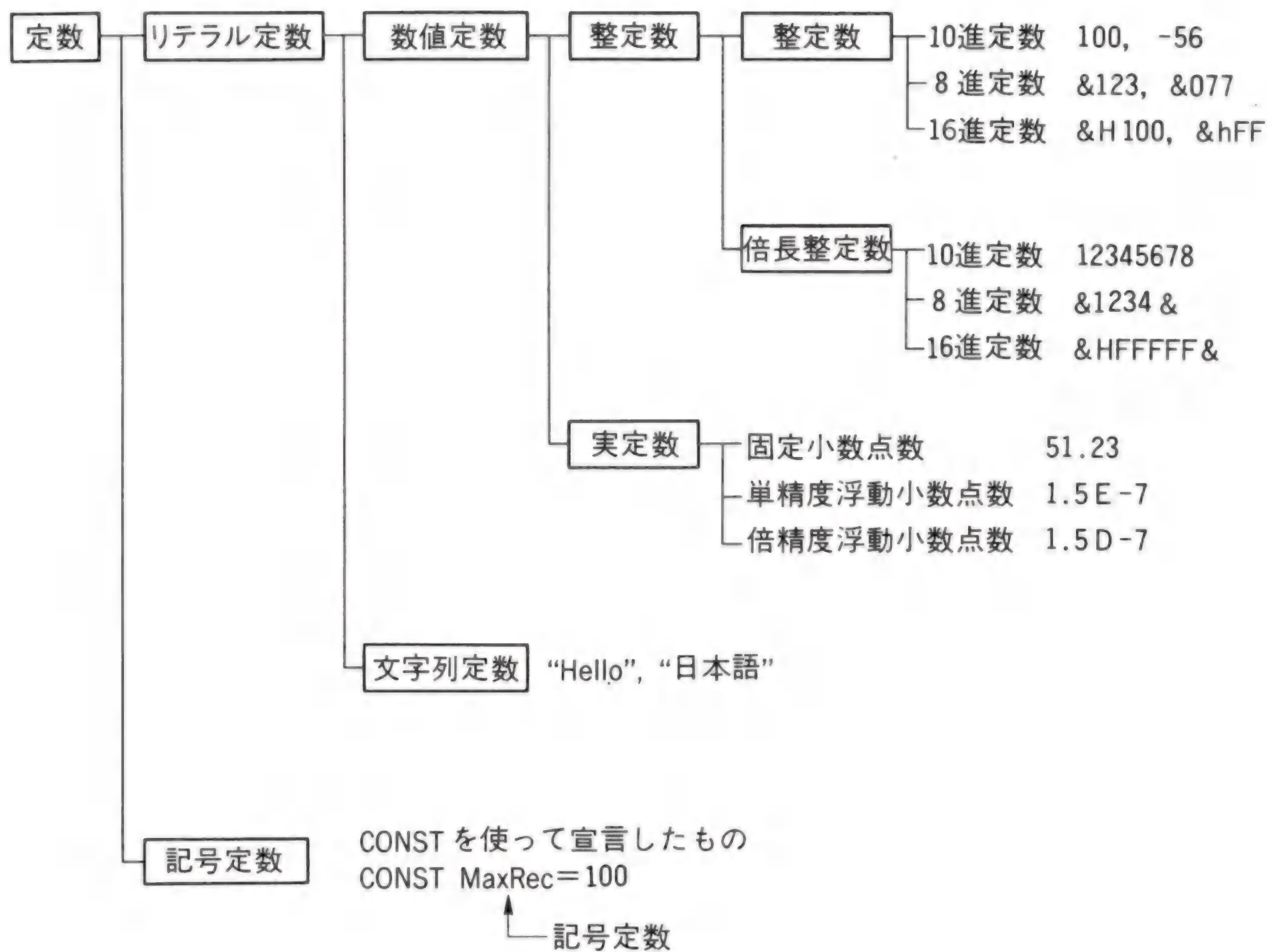
1 データ型の種類

Quick BASIC には次のようなデータ型があります。



2 定数

プログラムの実行中に変化しない値を定数といい，以下のように分類されます。



■リテラル定数

literal は「文字どおりの」という意味です。リテラル定数とは，100や“Hello”のような数値または文字列そのものを表す定数です。

リテラル定数は数値定数と文字列定数に別れます。数値定数の進数および精度は以下のような記号を用いて示します。

- ・ 8進数を示すには，先頭に&O，&o または&を付けます。
- ・ 16進数を示すには，先頭に&H または&h を付けます。
- ・ 8進または16進の倍長整定数を示すには，末尾に&を付けます。
- ・ 単精度浮動小数点数の指数を示す記号としてEまたはeを用います。
- ・ 倍精度浮動小数点数の指数を示す記号としてDまたはdを用います。

■記号定数

Quick BASIC では CONST 文により記号定数を宣言することができます。記号定数とは、定数に名前を割り当て、その名前を定数として扱えるようにしたものです。記号定数は次のように定義します。

CONST 記号定数名 = 値, 記号定数名 = 値...

記号定数のデータ型は、名前にその型を宣言する文字を付けなければ、割り当てられる値(式)のデータ型になります。

記号定数名の付け方のきまりは変数名の付け方のきまりと同じです。

例

CONST Boy=20, Girl=21, Seito=Boy+Girl

… Seito は $20+21=41$ と定義される。

CONST Pai # = 3.141592657#

…記号定数の末尾に付いた%, &, !, #, \$ は定数名の一部ではなくデータ型を示す。

3 変数

■名前のきまり

プログラムの中で使う値を格納するための箱(実際はメモリ)を変数といい、その箱に付ける名前を変数名といいます。

変数名は次の規則で付けます。

- ・変数名の先頭は英字で始まります。
- ・第2文字以後は英字、数字、ピリオド(.)が指定できます。
- ・変数名の長さは最大40文字です。
- ・予約語は使用できませんがそれを含んだものはかまいません。
- ・FN で始まる変数名は許されません。
- ・英大文字と英小文字は同じものとして扱われます。つまり、SUM, Sum, sum などとはすべて同じものとして扱われます。

記号定数、プロシージャ、関数、ラベルに付ける名前も、この規則に従います。

■変数の型宣言

CやPascalではプログラムで使用する変数は、必ず型宣言しなければなりません
が、BASICはこうした制約はなく、変数の型宣言をせずに使用した変数は単精度実
数型とみなされます。これを「暗黙の型宣言」といいます。

このような暗黙の型宣言によらず変数の型を明示的に宣言するには、次のような3
つの方法があります。

●変数名の後に型宣言文字を付加する

A\$, A%のように変数名の後に型宣言文字を付加して、その変数の型を宣言する
ことができます。型宣言文字として以下のものがあります。

%	(単精度)整数型	
&	倍長整数型	
!	単精度実数型	
#	倍精度実数型	
\$	文字列型	注) A%, A&, A!, A#, A\$はすべて異なる変数です。

●宣言文を使って宣言する

次のように宣言子と型名を指定して変数の型を宣言することができます。

宣言子	変数名, 変数名, ...AS	型名	
—COMMON	… 共用	—INTEGER	… (単精度)整数型
—DIM	… 配列	—LONG	… 倍長整数型
—REDIM	… 動的配列の再定義	—SINGLE	… 単精度実数型
—SHARED	… プロシージャ間の共用	—DOUBLE	… 倍精度実数型
—STATIC	… ローカル変数	—STRING	… 可変長文字列型
		—STRING * 長さ	… 固定長文字列型
		—タグ名	… ユーザ定義型 (レコード型)

例

```
DIM A AS STRING * 10
    ...変数 A を長さ10文字の固定長文字列型変数として宣言
DIM Sum AS INTEGER
    ...変数 Sum を整数型変数として宣言
```


●インプリシット宣言

DEFtyp 文により、変数の先頭文字で型を分類させることができます。

$\left\{ \begin{array}{l} \text{DEFINT} \\ \text{DEFLNG} \\ \text{DEFSNG} \\ \text{DEFDBL} \\ \text{DEFSTR} \end{array} \right\}$	アルファベット[-アルファベット], ...
---	------------------------

例

DEFINT I-N

...I~N で始まる変数(Jac, Man など) を整数型として扱うことを宣言

DEFDBL A, B, X-Z

...A, B, X~Z で始まる変数を倍精度実数型として扱うことを宣言

DEFINT A-Z

...すべての変数を整数型として扱うことを宣言

●宣言の優先順位

型宣言文字を付加した宣言と、宣言文を使った宣言は、DEFtyp 文に対し常に優先します。

たとえば,

```
DEFINT I-N
DIM Num AS LONG
```

のような宣言において、Num は倍長整数型として宣言されたことになります。

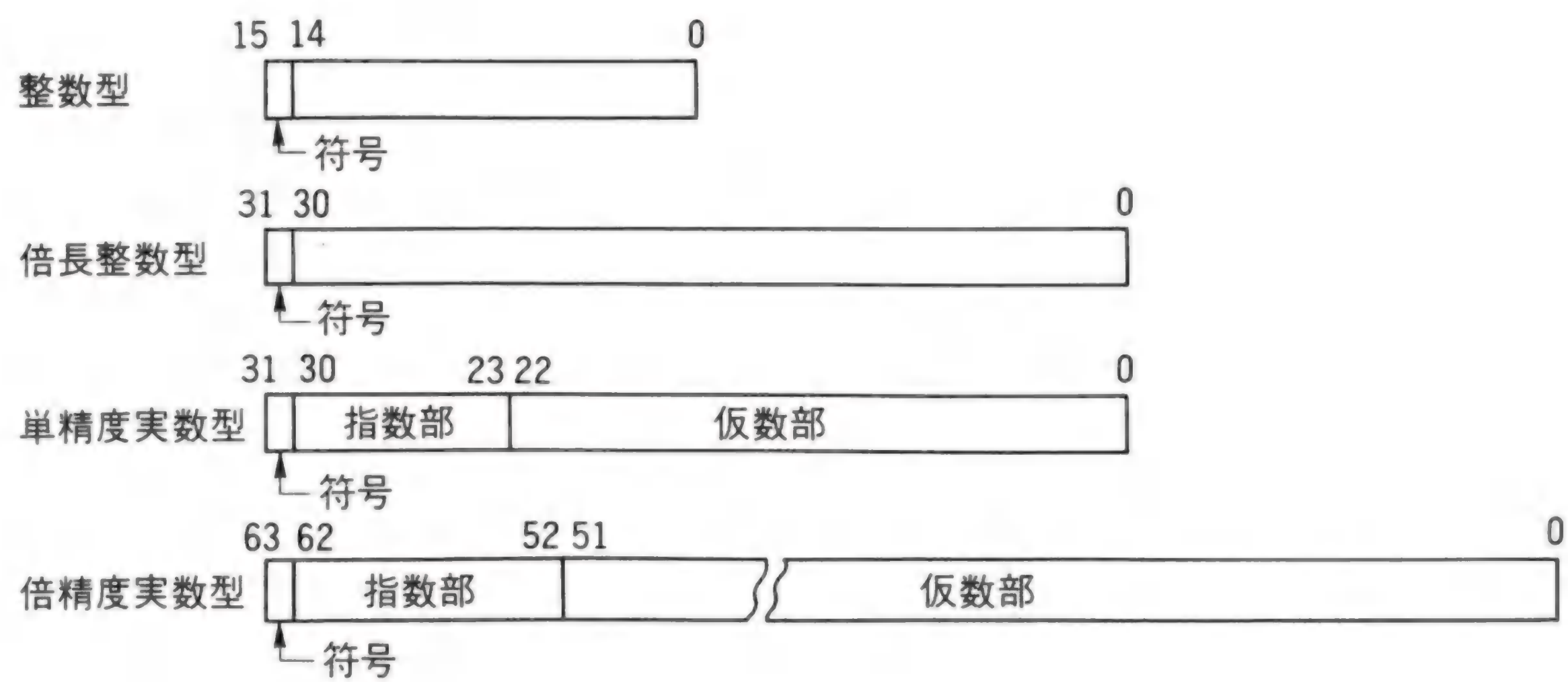
4 基本データ型

Quick BASIC において、あらかじめ定められた型(いわゆる基本データ型)のサイズとデータ範囲を以下に示します。

データ型	サイズ	扱える値の範囲	説 明
整数型	2バイト	-32768~32767	100や5という整数を扱う型
倍長整数型	4バイト	-2147483648 ~2147483647	100000のような有効桁の長い整数を扱う型
単精度実数型	4バイト	-3.402823E+38 ~3.402823E+38	3.14159のような小数点数を扱う型. 7 桁
倍精度実数型	8バイト	-1.797693134862316D+308 ~1.797693134862316D+308	3.1415927...のような有効桁の長い小 数点数を扱う型.16桁
文字型	可変長	最大32767文字	"Hello"のような文字を扱う型
	固定長		

注) E+38は10³⁸を意味します。
D+308は10³⁰⁸を意味します(D は Double の意味)。

整数型、倍長整数型、単精度実数型、倍精度実数型の内部データ形式を以下に示します。



■固定長文字列

一般に BASIC の文字列は可変長文字列で、文字列変数に入れる文字列の長さは自由自在です。Quick BASIC ではこのような可変長文字列のほかに、固定長文字列をサポートしています。

固定長文字列の宣言は、型指定子 `STRING` を用いて次のように行います。

```
DIM 変数名 AS STRING * 長さ
```

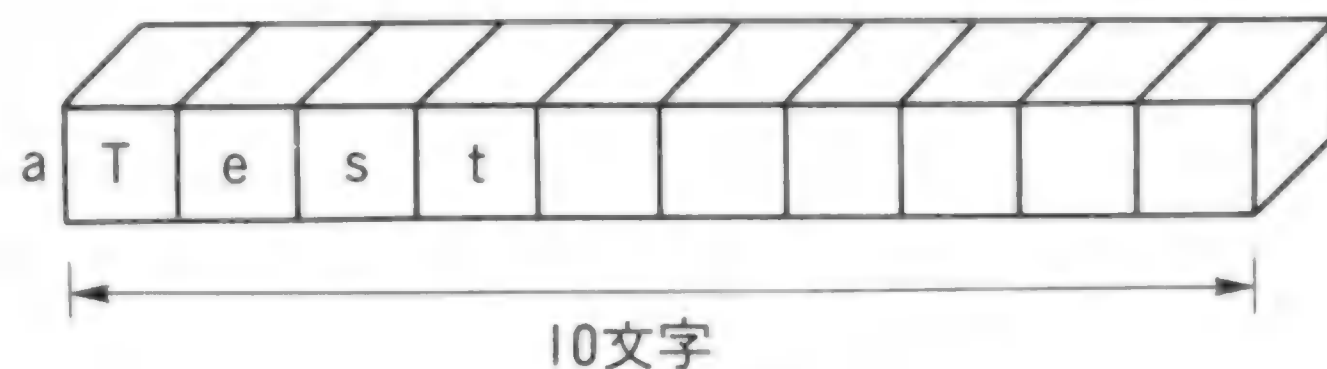
たとえば、

```
DIM a AS STRING * 10
```

と宣言すると、変数 `a` は10文字の文字を入れる固定長文字列変数となります。

```
a = "Test"
```

とすると、文字は次のように左づめで格納され、残った右は空白で埋められます。



したがって、

```
IF a = "Test" THEN
```

としても、この条件は一致しません。これは、

```
IF a = "Test" THEN
```

としなければなりません。

後述するレコード型のメンバには可変長文字列を指定できませんので、この固定長文字列を指定します。

固定長文字列変数に対し、指定された長さより長い文字列を代入すると、残った部分は捨てられてしまいます。

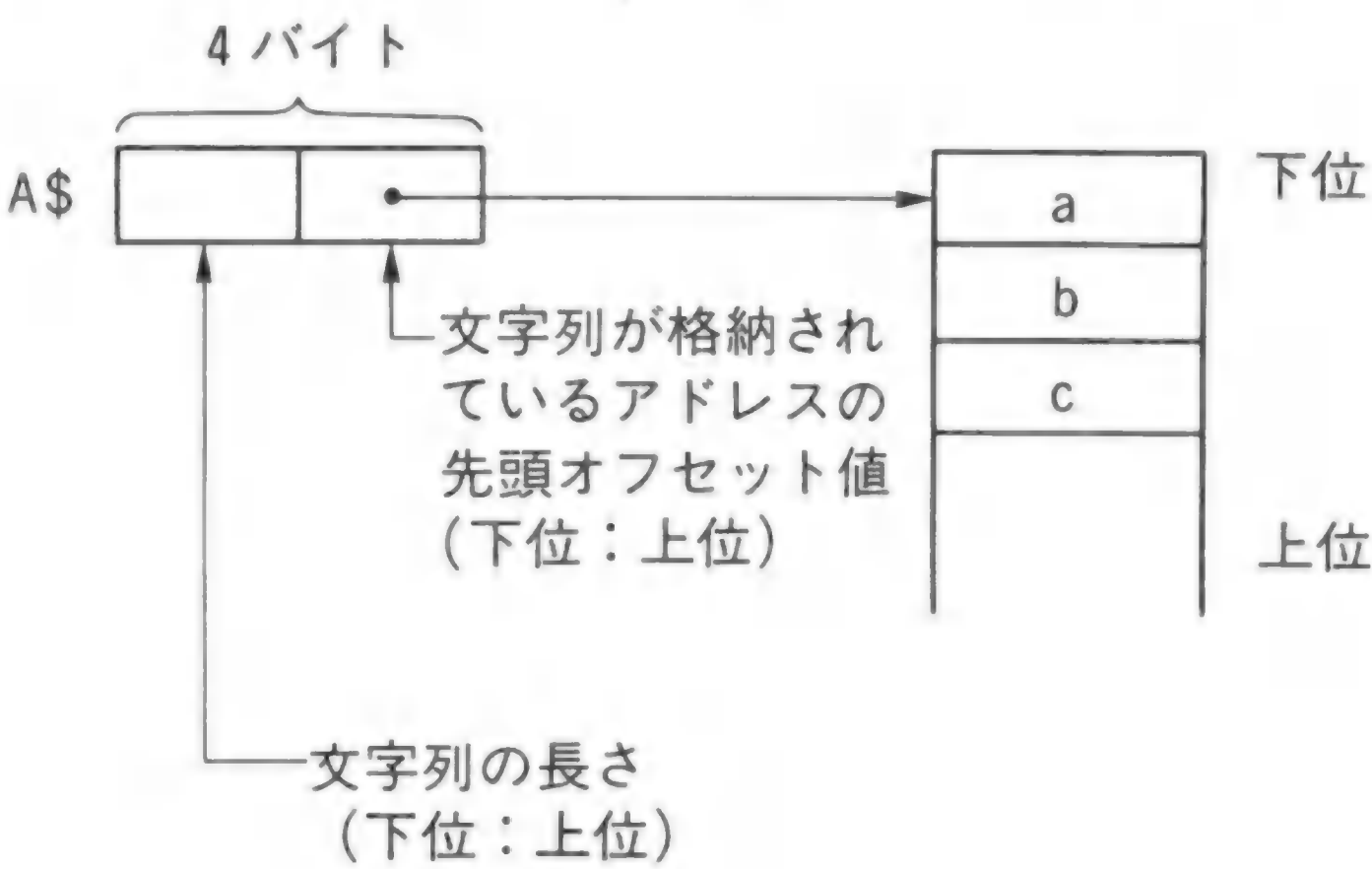
■文字列の格納イメージ

文字列データのメモリ上への格納イメージは、可変長文字列と固定長文字列とで異なります。

たとえば、

A\$= "abc"

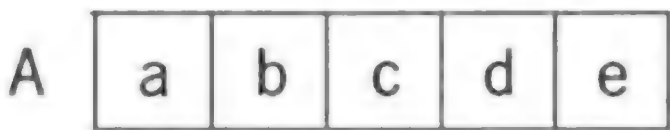
のような可変長文字列は次のようにメモリ上に格納されています。



つまり、可変長文字列変数は4バイトのサイズで、先頭2バイトで文字列の長さ、後部2バイトで、文字列が格納されているアドレスの先頭オフセット値を示しています。つまり、変数 A\$ がある場所と、実際に文字列が格納されている場所とが異なるわけです。

これに対し、固定長文字列では次のように変数の位置に文字列が格納されます。

```
DIM A AS STRING * 5
A = "abcde"
```



5 配列

配列は同種のデータを配列名と添字(要素番号)で管理するデータ構造の1つです。配列の添字が1つのものを1次元配列、2つのものを2次元配列、以下、3次元、4次元……と呼びます。

■配列の宣言

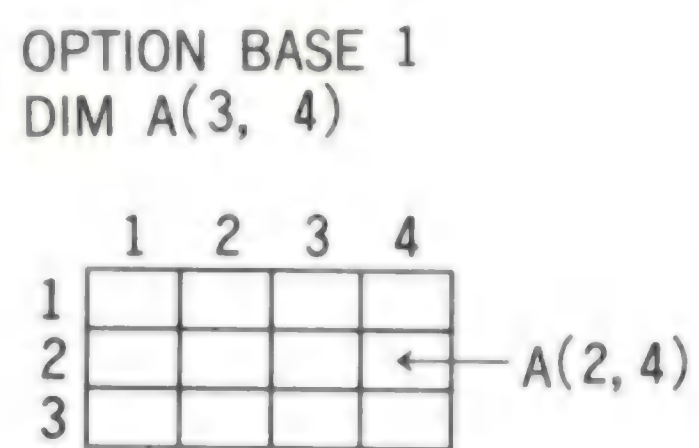
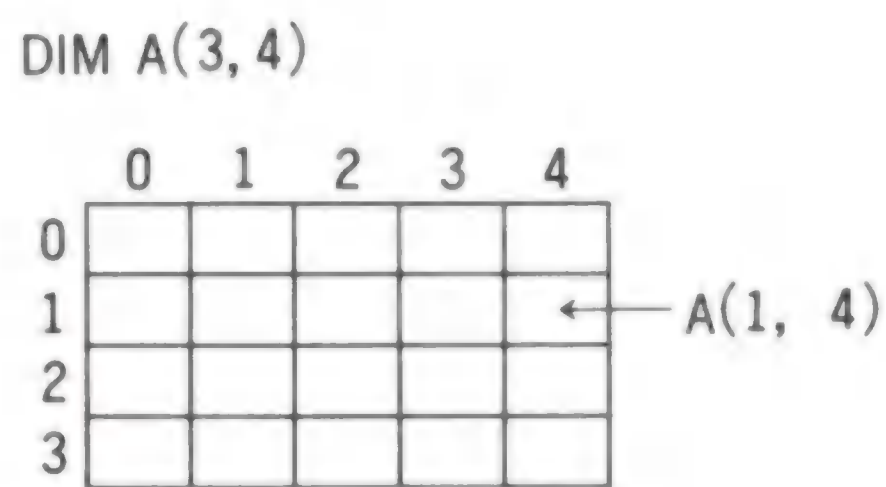
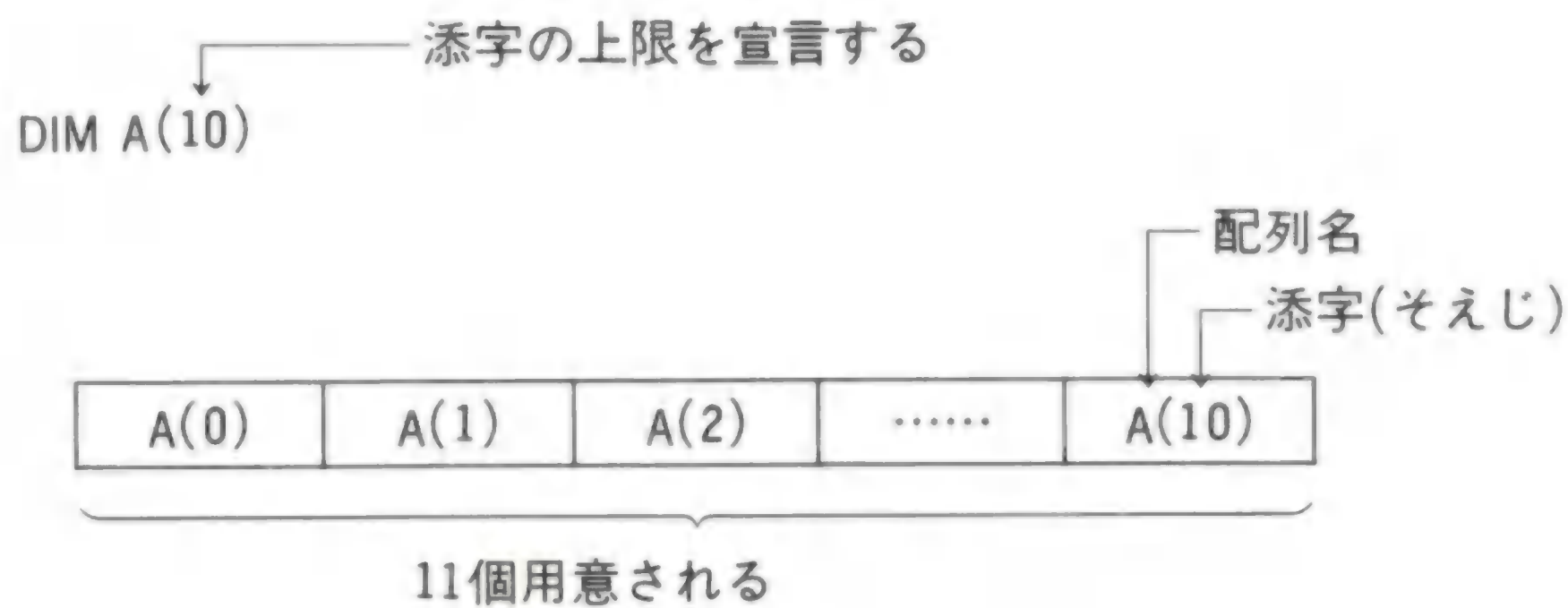
配列の宣言はDIMを用いて次のように行います。

```
DIM 配列名( 添字の最大値, 添字の最大値, ...), ...[AS 型名]
```

配列の要素は通常0番の要素から始まりますが、OPTION BASE 1と宣言することで1番の要素から始めることもできます。OPTION BASE 文はすべての配列に対して作用します。

「AS 型名」を省略すると、単精度実数型の配列とみなされます。

例



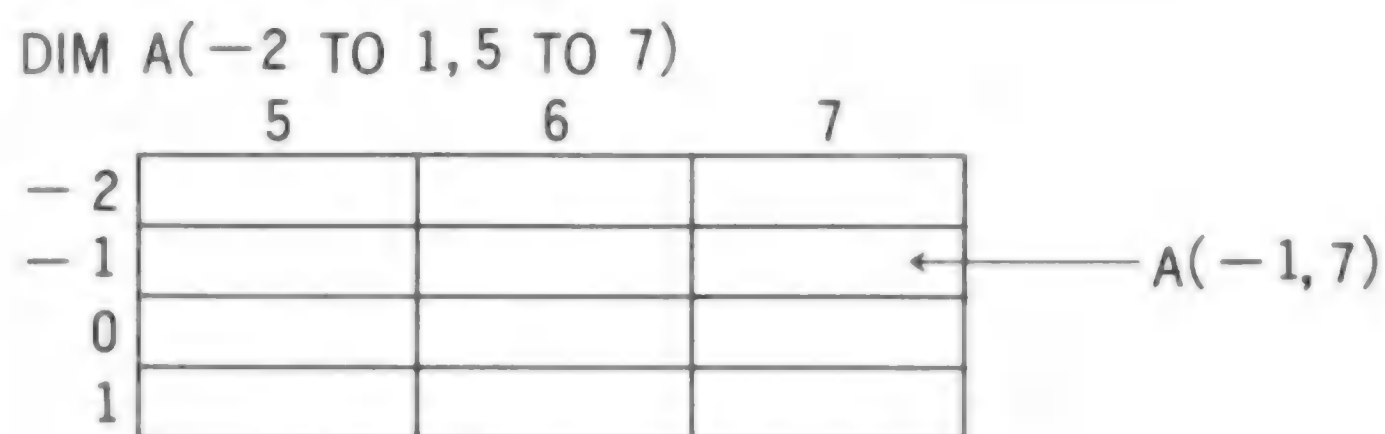
DIM Sum(100), A(3, 4) AS LONG
…配列 Sum(), A()は倍長整数型配列として宣言されます。

■部分範囲指定

Quick BASIC では配列の添字に部分範囲を指定することができます。

DIM 配列名(最小値 TO 最大値, 最小値 TO 最大値, …)

例



■配列における留意事項

配列の添字の範囲およびサイズについては以下のような制約があります。

- ・ 配列の添字には、-32768～32767の値が使えます。
 - ・ 1次元あたりの要素の数は最大で32768個までです。
 - ・ 1つの配列の次元の上限は60次元までです。
 - ・ 静的配列の最大サイズは64KB(キロバイト)までです。
- 注) 静的配列と動的配列の違いについては本章1-8を参照してください。

配列が必要とするメモリ量は、〈配列の要素数〉×〈その型のサイズ〉です。たとえば、

```
DIM A(255) AS LONG
```

という配列のサイズは、 $256 \times 4 = 1024$ バイト = 1KB(キロバイト)となります。
配列名は、単純変数の名前とは別物として扱われます。たとえば、

```
DIM A(100)
DIM A AS INTEGER
```

の A(0)～A(100)とAは別物です。しかし、このように配列名と単純変数名に同じものを使うとプログラムがわかりにくくなるので、このような使用はさけるべきです。

6 レコード型

配列は同じ型のデータをひとかたまりにしたものですが、レコード型は異なる型のデータをひとかたまりにしたものです。

下表のデータは、Galileo(ガリレオ) が1610年に発見した木星の4大衛星の衛星名、光度、周期に関するものです。

木星の4つの衛星

衛星名	光度〔等〕	周期〔日〕
Io	5	1.7691
Europa	6	3.5512
Ganymede	5	7.1545
Callisto	6	16.6890

1件のレコードとして扱う

こうしたデータは、衛星名、光度、周期をひとかたまりのデータとして扱えたら便利。このようにいくつかの項目をひとかたまりにしたものを「レコード」と呼び、各項目を「メンバ(フィールド)」と呼びます。レコード型はこのようなデータを定義するのに便利です。

■レコード型の定義

レコード型は TYPE~END TYPE 文を用いて次のように行います.

```
TYPE   タグ名
      メンバ名 AS  型名
      メンバ名 AS  型名
      ⋮
END   TYPE
```

これにより<タグ名>で示される新しい型が定義されたことになります. このようにユーザが定義する型を「ユーザ定義型」といいます.
各メンバを指定する型名は次のものです.

INTEGER	(単精度)整数型
LONG	倍長整数型
SINGLE	単精度実数型
DOUBLE	倍精度実数型
STRING *長さ	文字列型(固定長)

メンバに指定できる文字列型は, 固定長だけで, 可変長文字列は認められません.

例

先の表(木星の4つの衛星)のようなデータを扱うレコード型を定義するには次のようにします.

```
      ↙ タグ
TYPE   Eisei
      ↘ メンバの型
      Star   AS  STRING * 10
      Kodo   AS  INTEGER
      Shuki  AS  SINGLE
END↑TYPE
      ↳ メンバ
```

これにより, Eisei 型という新しい型が定義されたことになります.
Eisei をタグ, Star, Kodo, Shuki をメンバと呼びます. タグは変数を宣言するための金型のような働きをします.

■レコード型変数の宣言

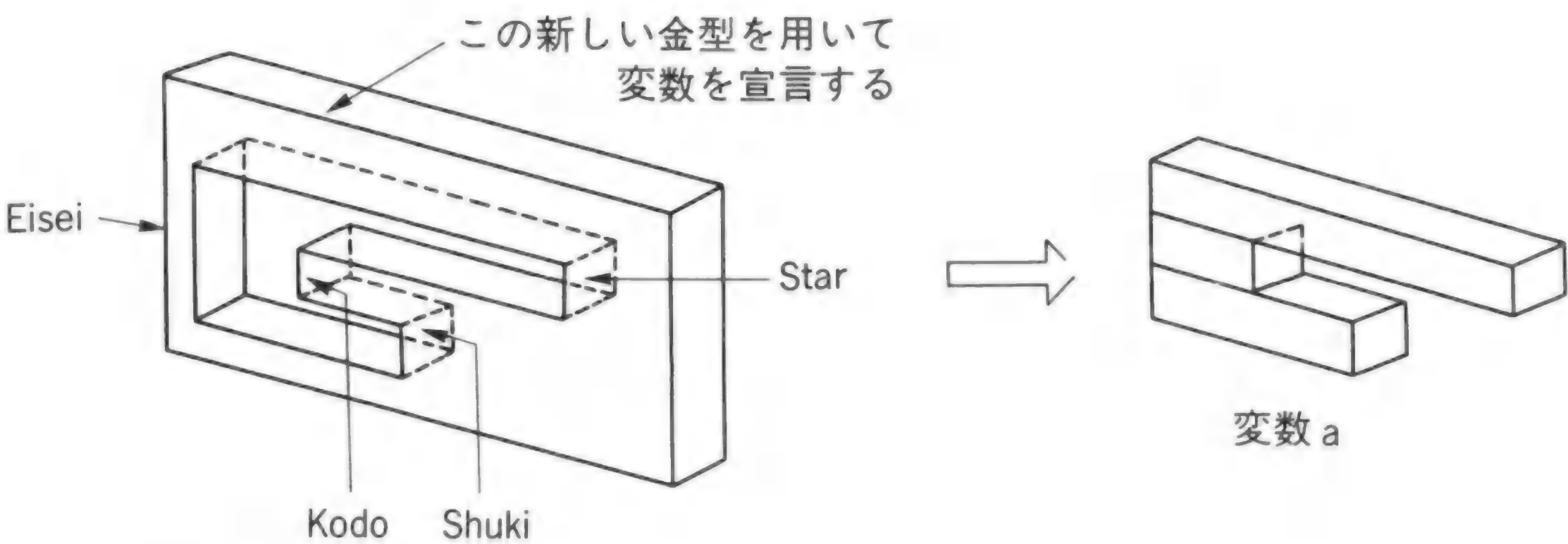
TYPE~END TYPE 文で定義したレコード型のタグ名を用いて、変数を宣言するには、

```
DIM 変数名, 変数名… AS タグ名
```

とします。

例

```
DIM a AS Eisei
… Eisei 型の変数 a を宣言
```



■メンバの参照

レコード型変数の各メンバを参照するにはピリオド(.)を用いて、

```
レコード型変数名.メンバ名
```

とします。たとえば、衛星 Io(イオ)のデータをレコード型変数 a に代入するには、次のようにします。

```
a.Star= "Io"  
a.Kodo= 4  
a.Shuki=1.7691
```

レコード型データは、ひとかたまりにして代入できますので、DIM b AS Eisei と宣言されているレコード型変数 b に対し、

```
b = a
```

とすることで、a の内容は一括で b に代入されます。

7 通用範囲(スコープ)

変数(あるいは記号定数)の内容を参照できる範囲を、スコープ(通用範囲：有効範囲)と呼びます。

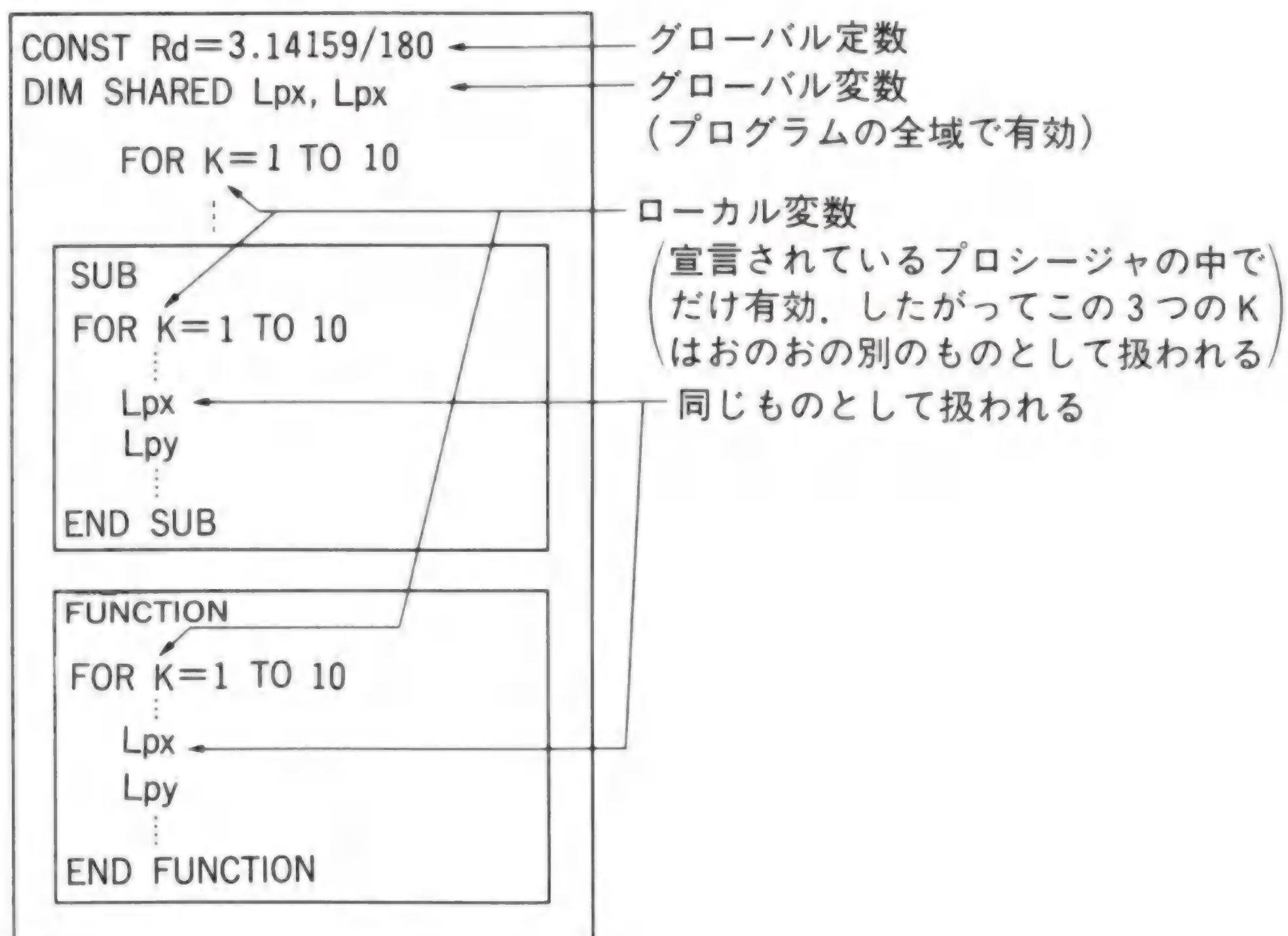
スコープは、グローバル(大域的)とローカル(局所的)の2種類に分けられます。グローバルなスコープはプログラム中のどの部分からも参照できますが、ローカルなスコープはそれが宣言(使用)されているプロシージャの中でだけ参照することができます。

■ローカル変数とグローバル変数

従来型 BASIC の変数はすべてグローバル変数でしたが、Quick BASIC ではグローバル変数とローカル変数の2種類が指定できます。

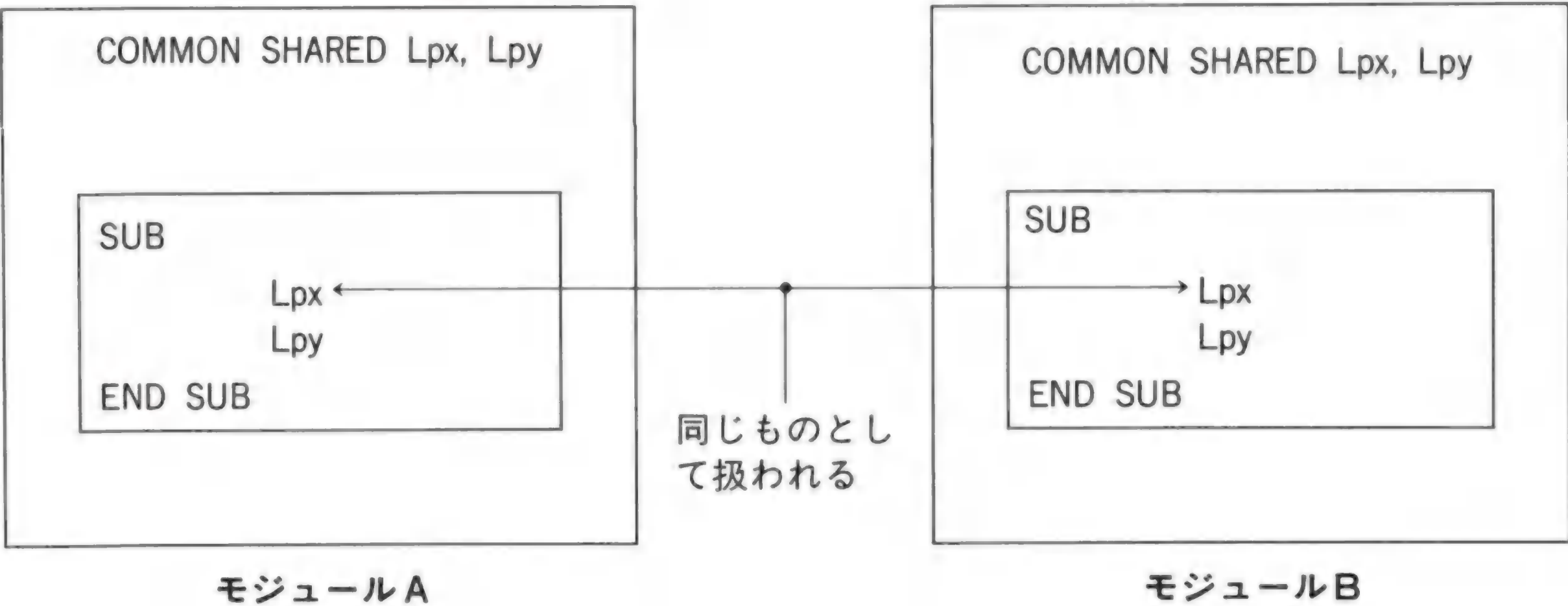
- ・一般的な変数はローカル変数です。ローカル変数は、それが宣言(使用)されているプロシージャの中でだけ有効です。
- ・メインルーチンにおいて、DIM SHARED で宣言された変数はグローバル変数です。グローバル変数はプログラムの全域で有効です。
- ・メインルーチンで宣言された記号定数はグローバル定数です。SUB あるいは FUNCTION プロシージャの中で宣言された記号定数はローカル定数です。

● プログラム



■モジュール間の変数の共有

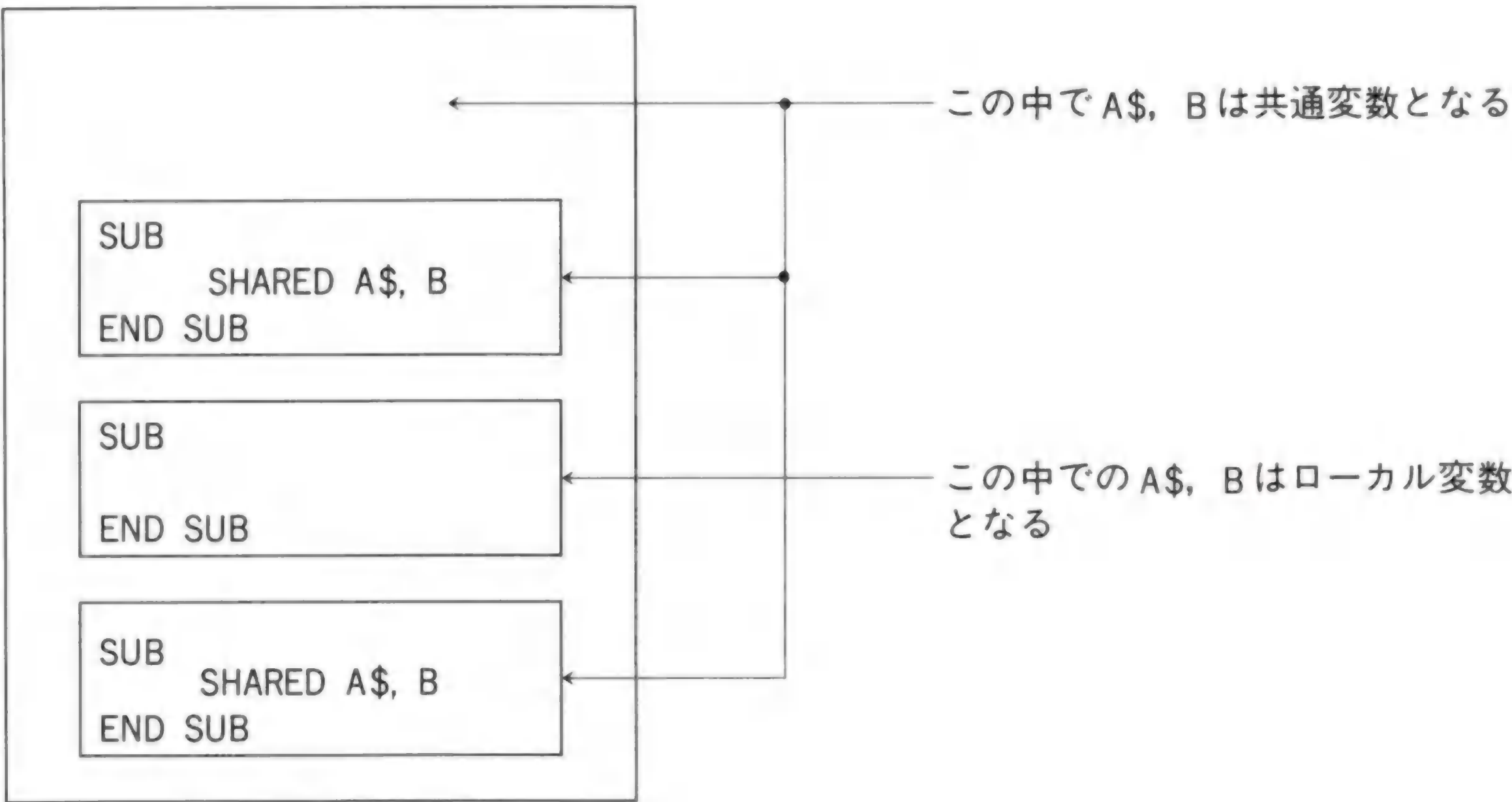
DIM SHARED で宣言された変数は、それが宣言されているモジュール内のすべてのプロシージャでグローバルです。つまり単一モジュール内での扱いとなります。これに対し、COMMON SHARED で宣言された変数は、異なるモジュールのすべてのプロシージャでグローバルです。



COMMON で変数を宣言する場合、変数の宣言順序は、すべてのモジュールの宣言において同一に行わなければなりません。したがって、COMMON の宣言文をインクルードファイルにしておき、各モジュールでは、このファイルをインクルードするようにすると安全です。

■プロシージャ間の変数の共有

SHARED 文 (DIM や COMMON のときの SHARED 指定とは異なる) を使えば、単一モジュール内の複数のプロシージャにおいて変数を共有することができます。



■ DEF FN 関数内のスコープ

DEF FN 関数はプロシージャではないので、その中で使用されている変数(引数は除く)は、メインルーチンの変数と同一のものです。DEF FN 関数内の変数をローカルにするには、STATIC 文を用います。

```
DEF FNAbc(x, y)
```

```
  STATIC i ←
```

```
    FOR i= 0 TO 10
```

```
      ⋮
```

```
END DEF
```

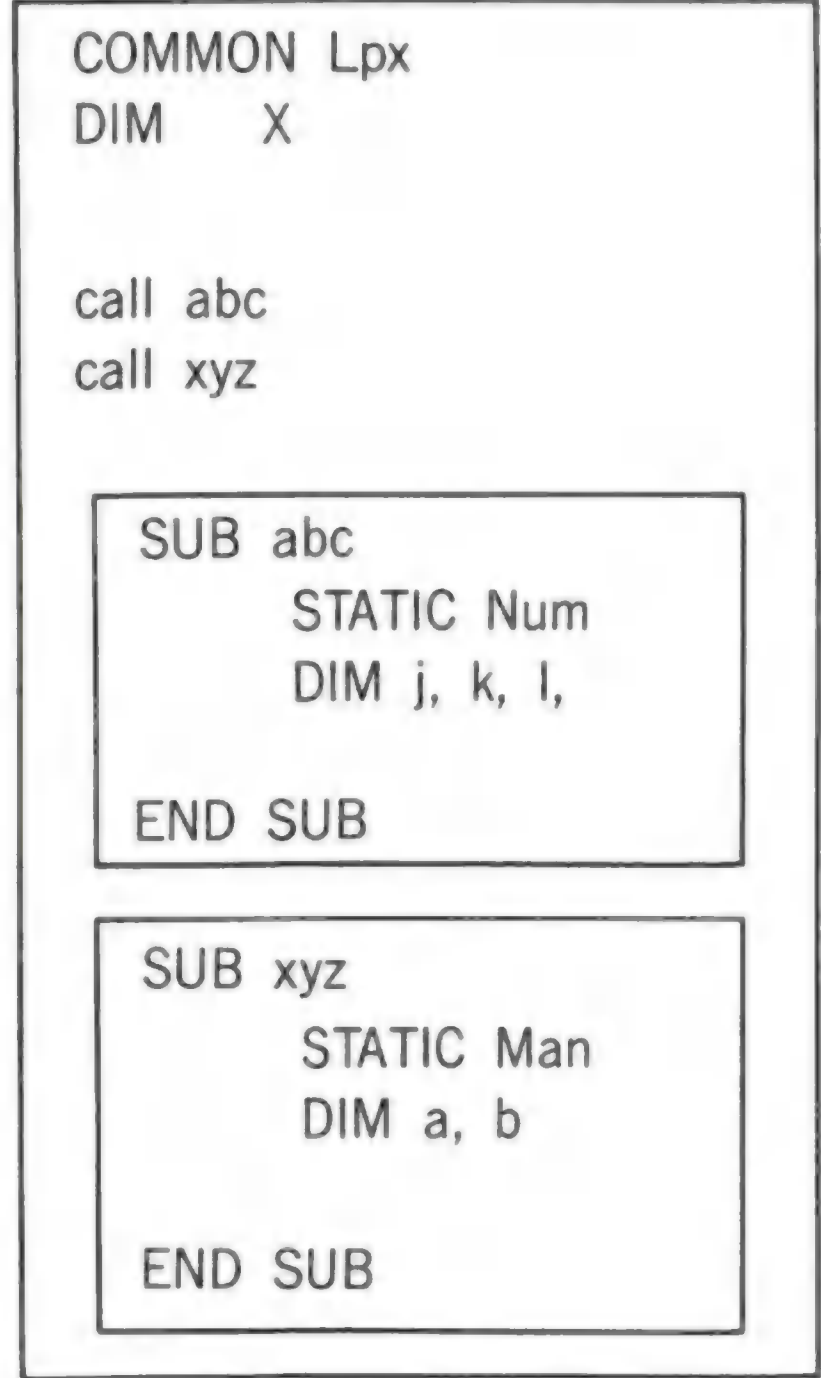
ローカル変数の指定

8 記憶クラス

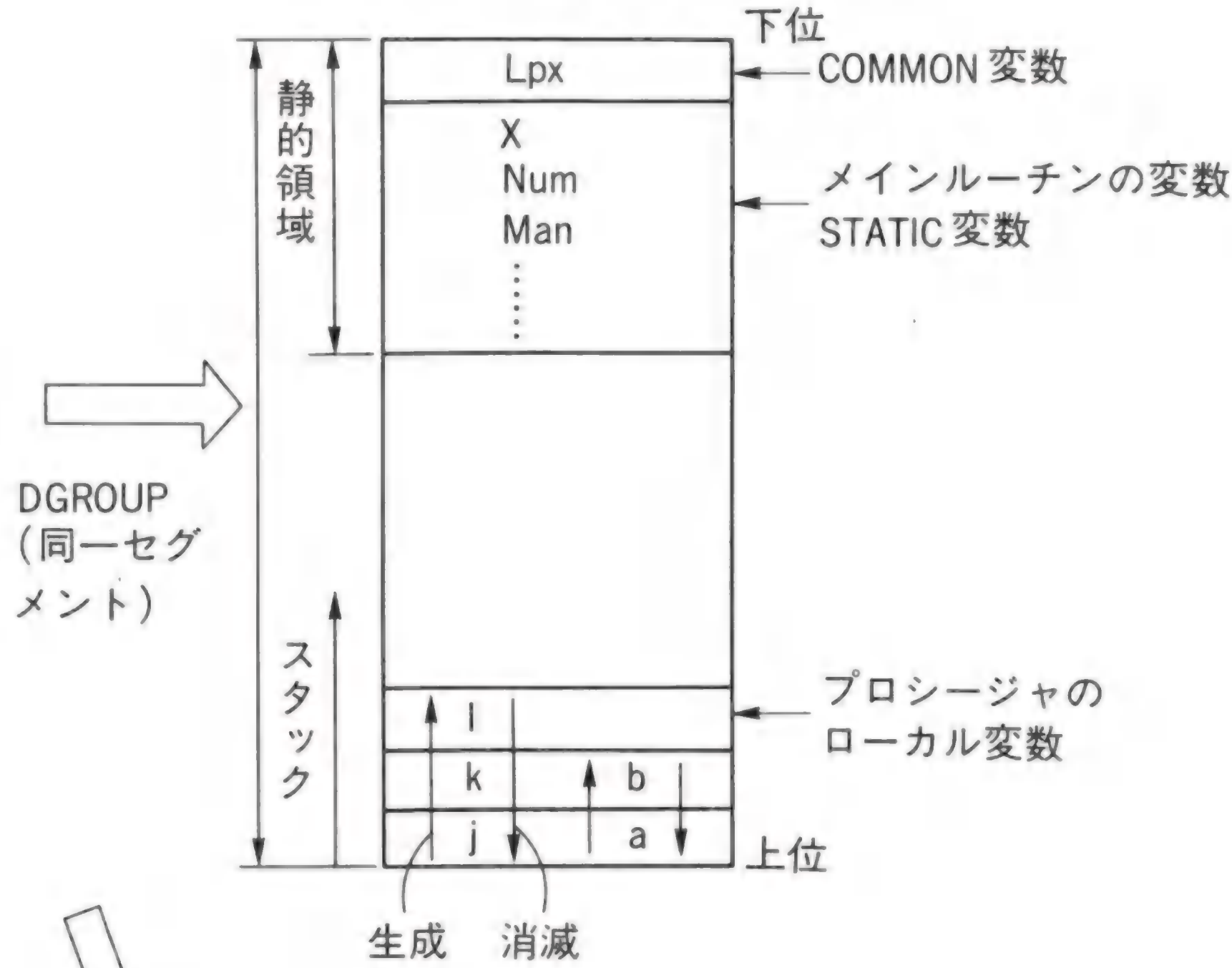
変数をメモリのどのような領域に割り当てるかで記憶クラスが決まります。メモリ領域は静的領域とスタック領域に大別されます。メインルーチンの変数や、プロシージャの中でSTATIC宣言した変数は、メモリの静的領域に格納され、プログラムの開始から終了まで同じアドレスに割り当てられています。これに対し、プロシージャのローカル変数は、メモリのスタック領域に割り当てられ、プロシージャの開始で生成され、プロシージャの終了で消滅します。

なお、変数のメモリ格納イメージは、.EXE ファイルにした場合と、QB 環境下の場合とでは下図のように多少異なりますので注意してください。

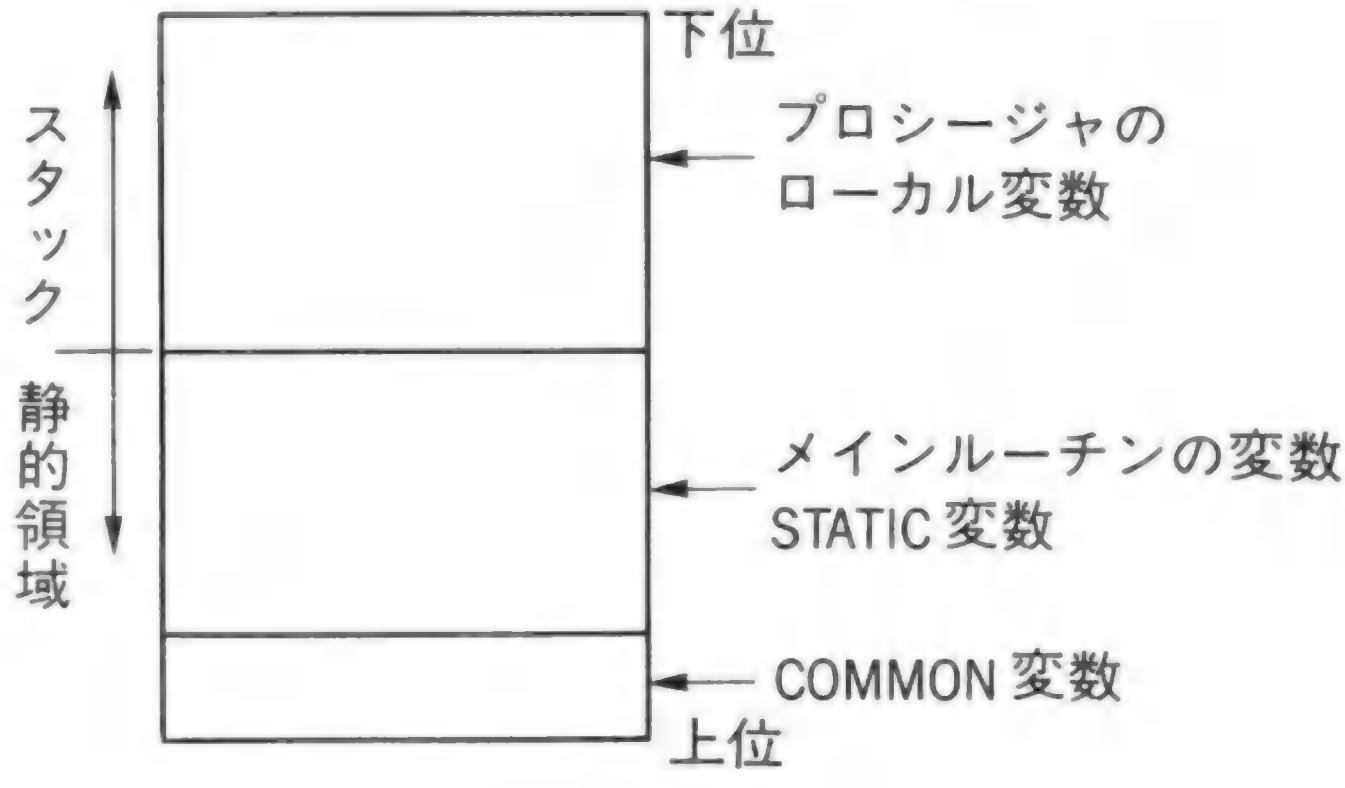
● プログラム



● メモリ格納イメージ(.EXE ファイル)



● メモリ格納イメージ(QB環境下)



■静的変数と自動変数

メモリの静的領域に割り当てられている変数を静的変数と呼び、メモリのスタックに割り当てられている変数を自動変数と呼びます。

静的変数は、プログラムの開始から終了まで同じアドレスに確保され、消滅することがないので、プロシージャへの再入時に、前の値を保存していることになります。

自動変数は、それが宣言(使用)されているプロシージャが呼び出されたときにゼロまたはヌル(空文字列)に初期設定されます。

次の例は静的変数を用いて、あるイベントが起こった回数をカウントするものです。

```
⋮  
IF  ある条件    THEN  call  Count  
⋮  
  
SUB  Count  
    STATIC  Num  
    Num =Num + 1  
    ⋮  
END  SUB
```

Num は STATIC 変数なので、Count が呼ばれるたびに前の Num の値を + 1 していきます。もし、Num を STATIC 指定しなければ、Count が呼ばれるたびに Num は 0 に初期設定されてしまいます。

なお、次のようにプロシージャ全体に STATIC 属性を与えることで、すべてのプロシージャ内の変数を STATIC 変数にすることもできます。

```
SUB  プロシージャ名(引数, ...)  STATIC  
⋮  
END  SUB
```


■動的配列

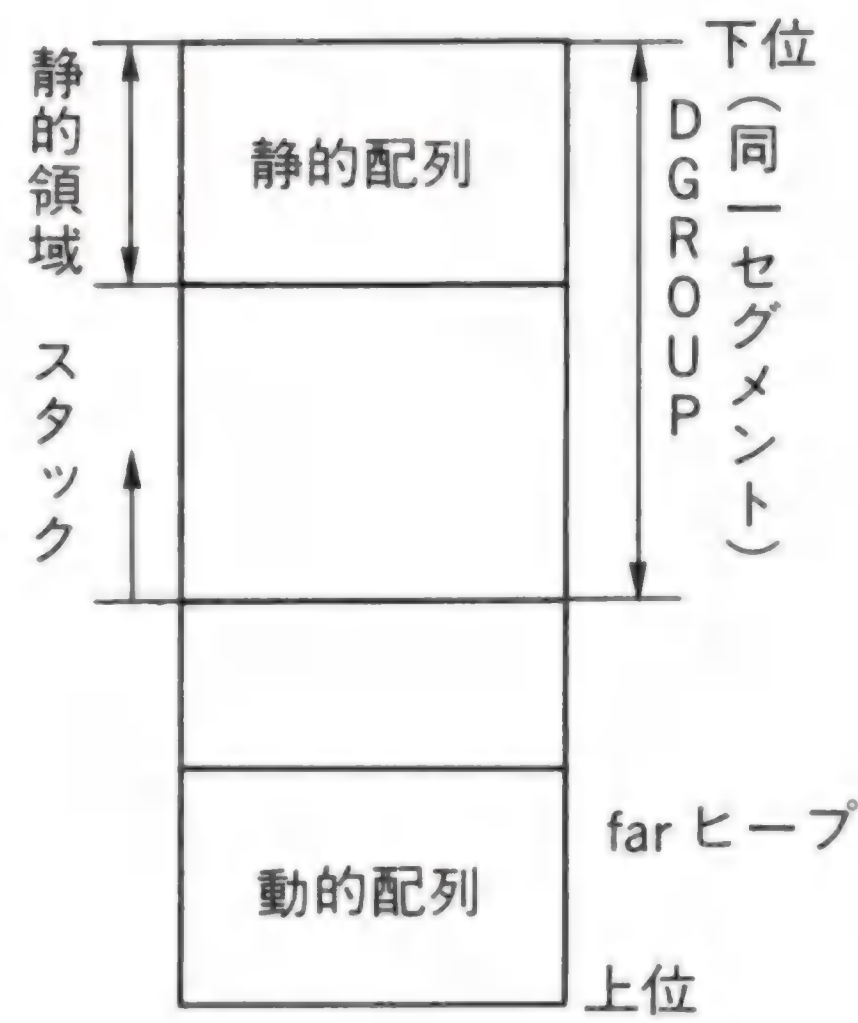
プログラムのコンパイル時に記憶領域を与えられる配列を静的配列と呼び、プログラム実行時に領域を与えられる配列を動的配列と呼びます。

宣言時に定数の添字を指定したものが静的配列になり、変数の添字を指定したもの、または最初に COMMON 文で宣言されている配列は動的配列になります。

```
DIM a(100) ← 静的配列

N=50
DIM b(N) ← 動的配列

COMMON c( )
DIM c(50) ← 動的配列
```



原則的には、静的配列は DGROUP セグメントの静的領域にとられ、動的配列は far ヒープ領域にとられますが、配列のメモリ割り当ては、EXE 形式にした場合と、QB 統合開発環境下の場合とでは以下のように異なります。

● EXE 形式の場合

- ・動的配列は far ヒープ領域
- ・静的配列は DGROUP
- ・可変長文字列の配列は静的配列、動的配列とも DGROUP

● QB 環境下

- ・COMMON 静的配列、すべての可変長文字配列は DGROUP
- ・これ以外の配列は far ヒープ領域

なお、配列を動的配列にするか静的配列にするかは、\$DYNAMIC、\$STATIC メタコマンドによっても指定することができます。

9 混合演算と型変換

数値データは、異なる型同士でも混合して演算することができます。こうした混合演算では精度の高い型に変換されて演算が行われます。たとえば、

```
A#=10#/3
```

は、10#/3#として演算が行われます。

なお、型変換は式の中の各演算単位で行われていくので、

```
x%=1 : y!=3 : z#=3  
PRINT x% / y! * z#
```

のような式では、まず x% / y! が単精度実数型として演算され、その結果が倍精度に変換され z# と乗算されますから、結果は1.0 とはなりません。

また、数値データが異なる型の変数に代入されるときは、次のような規則で変換されます。

- ・ **整数型変数←実数データ**

実数データが整数に変換される場合は、小数点以下が四捨五入されます。なお、実数データの整数部が-32768～32767の範囲を越えていればエラーが起こります。

```
A%=34.4      A%は34  
B%=34.5      B%は35
```

- ・ **実数型変数←整数データ**

整数データが実数型に変換される場合は、.0 が付加されます。

- ・ **単精度実数型変数←倍精度実数データ**

倍精度実数データが単精度実数型に変換される場合は、有効桁数が7桁に丸められます。

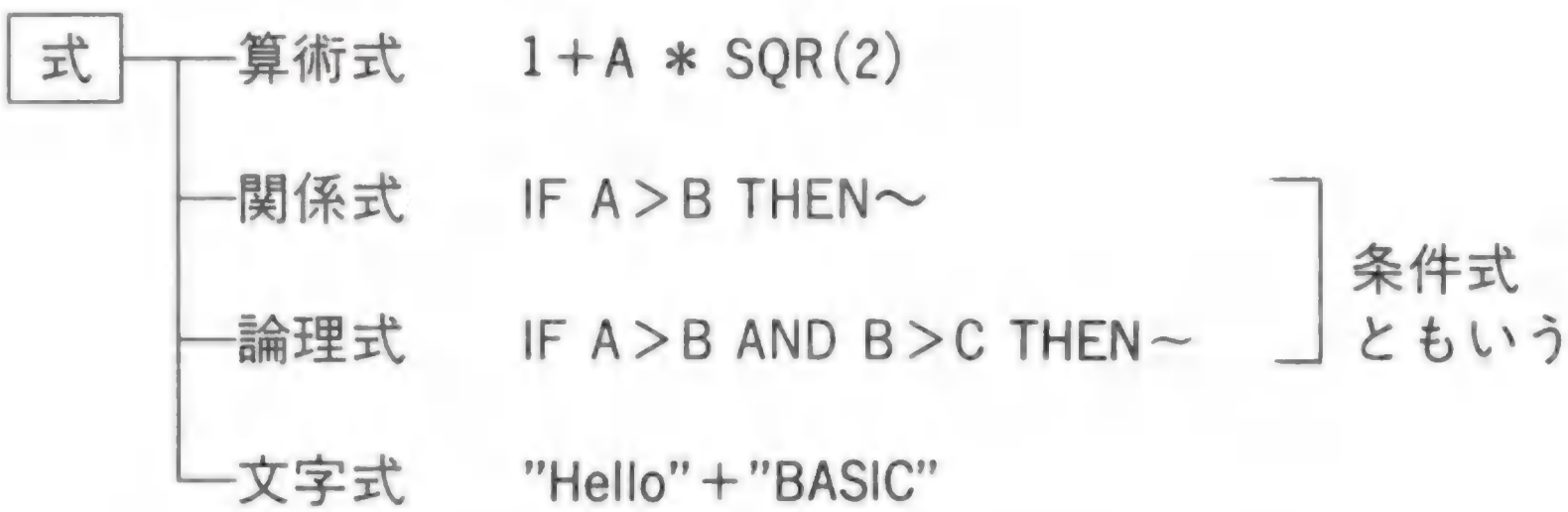
4-3

式と演算子

1

式

定数、変数、関数、およびこれらを演算子で結び付けたものを式といい、次のような種類があります。



2

演算子の種類と優先順位

Quick BASIC の演算子の種類と優先順位を以下に示します。

	演 算 子	優先順位
	かっこで囲まれたもの 関数	1 2
算術	^ - (負記号) * , / ¥ MOD + , -	3 4 5 6 7 8
関係	= < > < = > > = > < = < > =	9
論理	NOT AND OR XOR EQV IMP	10 11 12 13 14 15

これらの演算子は算術演算子、関係演算子、論理演算子に分類されます。

3 算術演算子

算術演算子として以下のものがあります。

● 算術演算子

演算子	意 味	優先順位
^	べき乗	3
-	負符号	4
*	乗 算	5
/	除 算	5
+	加 算	8
-	減 算	8
¥	整数の除算	6
MOD	整数除算の余り	7

例

- 10 ¥ 3

…整数型の除算を行うので結果は 3
- 10 MOD 3

…10÷3の余りで 1
- 10 ¥ 3 * 3

…¥より*のほうが優先順位が高いので3 * 3が先に計算され、10¥9となるので結果は 1

4 関係演算子

条件判断の際に大きい、小さい、等しいなどの大小関係を判定する演算子を関係演算子と呼び次の6つがあります。

● 関係演算子

演 算 子	意 味	優先順位
=	等しい	9
<> または ><	等しくない	
<	小さい	
<= または =<	小さいか等しい	
>	大きい	
>= または =>	大きいか等しい	

比較の結果は、条件を満たせば式の値は真(−1)、満たさなければ偽(0)となります。

一般に IF 文の条件式は、条件式の値が 0 なら偽、非 0 なら真として判断されます。

実数型のイコール比較は誤差を伴うので避けてください。たとえば、

```
Sum=0.0
100 Sum=Sum +0.1
    IF Sum=100.0 THEN 200
    GO TO 100
200
```

のようなプログラムにおいて、Sum の値が正確に100.0になる保証はありませんから、IF 文の条件式が一致しない可能性があります。

5 論理演算子

いくつかの条件式を組み合わせるものを論理演算子と呼び、次の 6 つがあります。

● 論理演算子

演算子	意 味	優先順位
NOT	否定	10
AND	かつ	11
OR	または	12
XOR	排他的論理和	13
EQV	同値	14
IMP	包含	15

T(TRUE)を真、F(FALSE)を偽を示すものとして、X と Y の論理演算を行ったときに得られる結果を表にしたものを以下に示します。

● 真理値表

		NOT	X AND	X OR	X XOR	X EQV	X IMP
X	Y	X	Y	Y	Y	Y	Y
F	F	T	F	F	F	T	T
F	T	T	F	T	T	F	T
T	F	F	F	T	T	F	F
T	T	F	T	T	F	T	T

例

`0 <= A AND A <= 100`

… A が 0 ~ 100 の範囲に入るかの判定

`A < B OR A < C AND A > 10`

… AND のほうが優先順位が高いため A < C AND A > 10 が先に判定される。

■ビット演算

NOT~IMP の論理演算子は内部的にはビット演算を行っています。

また関係式は比較の結果が真なら -1, 偽なら 0 となります。たとえば, A > B が真(-1 : &HFFFF), B > C が偽(0 : &H0000)の場合, A > B AND B > C は &HFFFF AND &H0000 が行われ, ビットごとの AND がとられて結果は &H0000 (0), つまり, 偽となります。

条件式の場合は, このような内部的なビット演算は考えず, 単に真/偽だけを考えればよいわけですが, NOT~IMP の論理演算子を用いて意図的にビットごとの論理演算を行うこともできます。

ビット演算の真理値表は, 先の真理値表の F を 0, T を 1 に読み換えたものになります。

● NOT (否定: ビット反転)

NOT は各ビットを反転します。

たとえば, NOT &hffla は &h00e5 となります。

NOT)	1111	1111	0001	1010	…&hffla
		0000	0000	1110	0101	…&h00e5

● AND (論理積)

AND は特定ビットをマスクするときによく使われます。

たとえば, n AND &h7fff は n の最上位(第15)ビットを強制的に '0' にします。

		1111	1111	0001	1010	…n
AND)	0111	1111	1111	1111	…&h7fff
		0111	1111	0001	1010	
		↑				
		0				0 にマスクされた

● OR（論理和）

OR は特定ビットを立てるときによく使われます。
たとえば、n OR &h000f は n の下位 4 ビットを強制的に‘1’にします。

	1111	1111	0001	1010	…n
OR)0000	0000	0000	1111	…&h000f
	1111	1111	0001	1111	

↑
1 が立った

● XOR(排他的論理和)

XOR は特定ビットを反転するときによく使われます。
たとえば、n XOR &h000f は n の下位 4 ビットをビット反転します。

	1111	1111	0001	1010	…n
XOR)0000	0000	0000	1111	…&h000f
	1111	1111	0001	0101	

↑
ビット反転

6 文字列演算

文字列演算として、連結と比較が行えます。
文字列は演算子+によって連結することができます。

A\$= “Hello”+“ Basic” → A\$ は “Hello Basic”
A\$= “Hello” : B\$= “Basic”
C\$=A\$+“_”+B\$ → C\$ は “Hello Basic”

文字列の大小はアルファベット順(辞書順)に行われます。正確にいうなら、文字の大小はその文字のアスキー・コードの大小で区別されます。2つの文字列を比較していて、文字列の片方が短くて比較が途中で終わった場合は、短い文字列のほうが小さくなります。
たとえば、次のような大小関係になっています。

“A” < “AA” < “AB” < “B” …… “Z” < “a”

1つの文字定数および文字変数で扱うことのできる文字列の長さは32767文字以下です。

Quick BASIC では、固定長文字列をサポートします。固定長文字列の比較は注意が必要です。

```
DIM A AS STRING * 10  
A = "/"  
IF A = "/" THEN ~
```

のような比較ではAの内容が、

/									
---	--	--	--	--	--	--	--	--	--

となっているので、“/”と比較しても一致しません。

4-4

制御構造

1 構造化プログラミングと近代的流れ制御構造

構造化プログラミングは、オランダのダイクストラ (E. W. Dijkstra) が提唱したプログラミングの方法論の 1 つです。

ダイクストラは、すべてのプログラムは、

- 接続(sequence)
- 判断 (if then else)
- 前判定反復 (while)

という 3 つの基本制御構造を組み合わせて書くことができるとしました。これをストラクチャード定理といいます。

しかし、実際にプログラミングする場合は、

- 後判定反復 (do～loop)
- 複数条件判断 (select case)
- 所定回反復 (for)

なども必要になります。以上の 6 つを近代的流れ制御構造と呼びます。

なお、接続というのは単に実行文が上から下に並んでいるものをいい、特別な制御文があるわけではありません。

従来型 BASIC は、後判定反復、複数条件判断などの制御構造がありませんでしたが、Quick BASIC では近代的流れ制御構造を完備しています。さらにブロックという概念の導入により、IF 文が構造的にスマートに記述できるようになりました。

2 判断

条件によって分岐するもので、単純 IF 文、ブロック IF 文、SELECT CASE 文があります。

■単純 IF 文（従来型 IF 文）

次のようにすべてを 1 行に書き、最後に END IF を置かない形式の IF 文を単純 IF 文(従来型 IF 文)と呼びます。

IF 条件式 THEN 文 1 ELSE 文 2

文はマルチステートメントで区切れば複数書けますが、1 行に収まる範囲です。

■ブロック IF 文

THEN 節、ELSE 節にブロックを置くことができる IF 文をブロック IF 文といい、Quick BASIC で追加された強力な制御構造です。

いくつかの文を文法的にひとかたまりにしたものをブロック(複文)といいます。

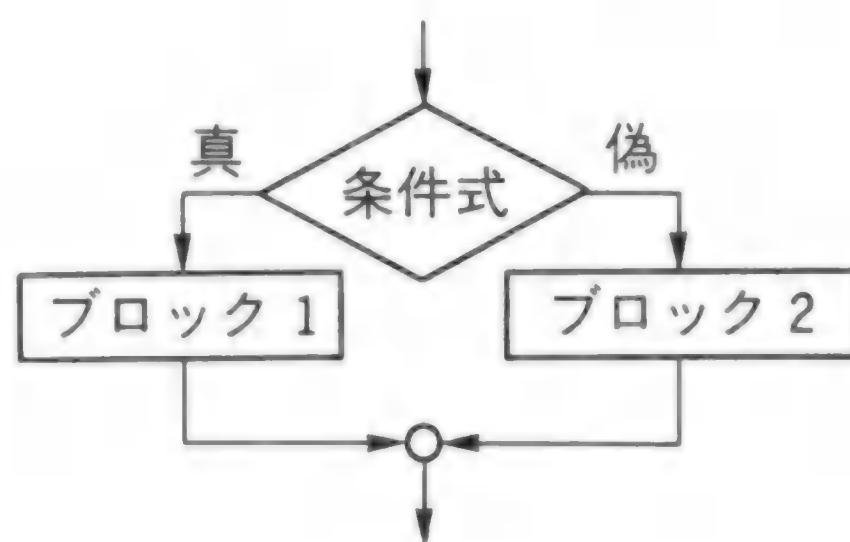
IF 条件式 THEN

ブロック 1

ELSE

ブロック 2

END IF



次のように ELSEIF 節を置くことで、複数の条件判断を置くことができます。

IF 条件式 1 THEN

ブロック 1

ELSEIF 条件式 2 THEN

ブロック 2

ELSEIF 条件式 3 THEN

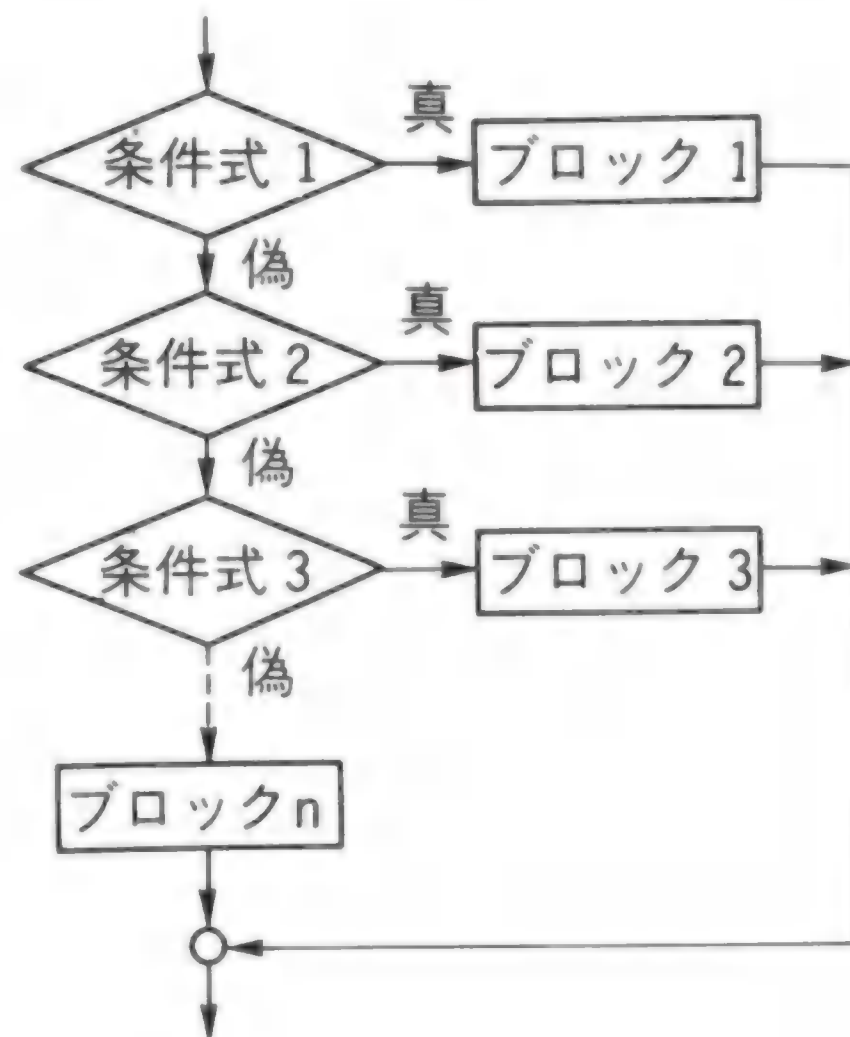
ブロック 3

⋮

ELSE

ブロック n

END IF

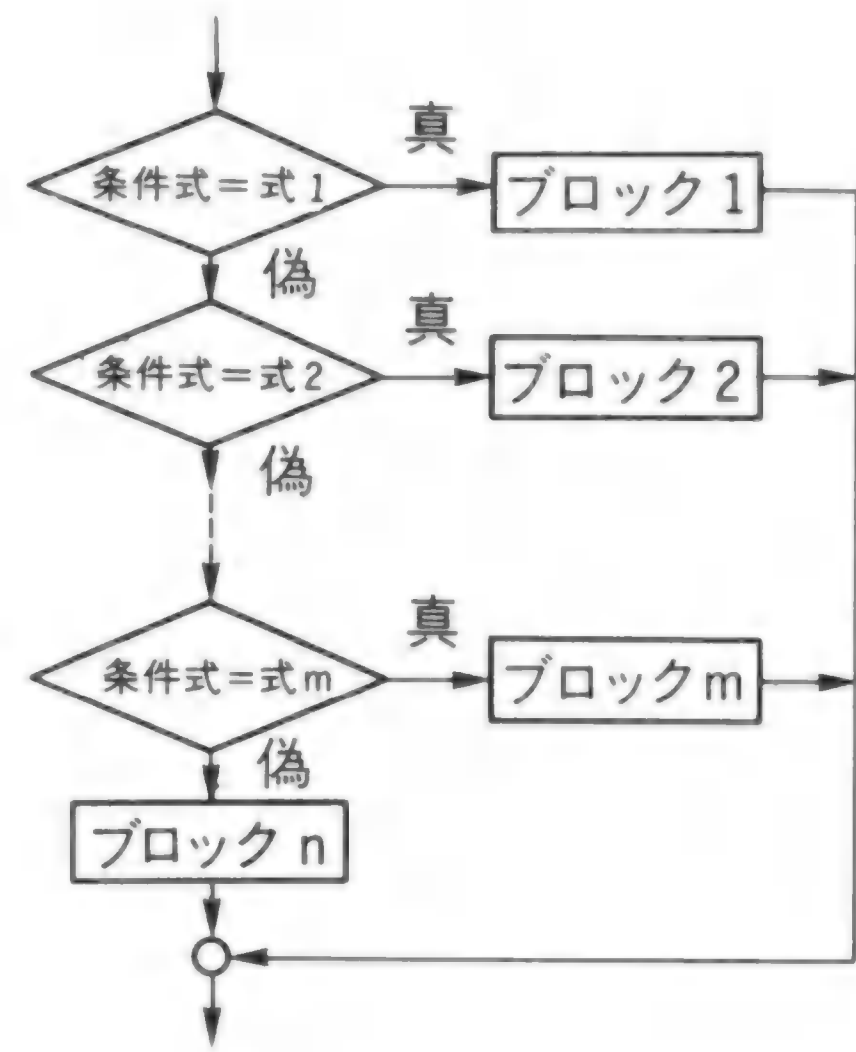


ELSEIF 節や ELSE 節は置くものがなければ省略できますが、END IF は省略できません。

■複数条件判断（SELECT CASE 文）

ブロック IF の ELSE IF 節により、複数の条件を判定することができましたが、SELECT CASE 文を用いればもっとスマートな複数条件判定をすることができます。

```
SELECT CASE 条件式
CASE 式 1
    ブロック 1
CASE 式 2
    ブロック 2
    ⋮
CASE 式 m
    ブロック m
CASE ELSE
    ブロック n
END SELECT
```



〈条件式〉と一致する式を〈式 1〉から順に探し、一致したところのブロックを実行して SELECT CASE 文から抜けます。もし一致する式がなければ CASE ELSE 節のブロックが実行されます。CASE ELSE 節はなくてもかまいません。

CASE の後に書く式は、5 や“abc”のような値のほかに、次のように値の範囲を指定することができます。

```
CASE 式 1 TO 式 2
CASE IS 関係式
```

3 くり返し

くり返しを行う制御文として、所定回反復、前判定反復、後判定反復があります。

■所定回反復（FOR 文）

くり返し回数があらかじめ定まっているくり返しを行います。

```
FOR ループ変数 = 初期値 TO 最終値 STEP きざみ値
```

文
⋮

```
NEXT ループ変数
```


■前判定反復（WHILE～WEND 文，DO～LOOP 文）

前判定反復は，くり返し回数が定まっていないくり返しを行います。そして，くり返しの終了判定をループの先頭で行います。

前判定反復を行う制御文として DO WHILE～LOOP 文，DO UNTIL～LOOP 文，WHILE～WEND 文の 3 つがあります。

WHILE～WEND と DO WHILE～LOOP は同じ機能です。WHILE～WEND は従来型 BASIC にあったもので，DO～LOOP は Quick BASIC で新たに導入したものです。

WHILE 条件式

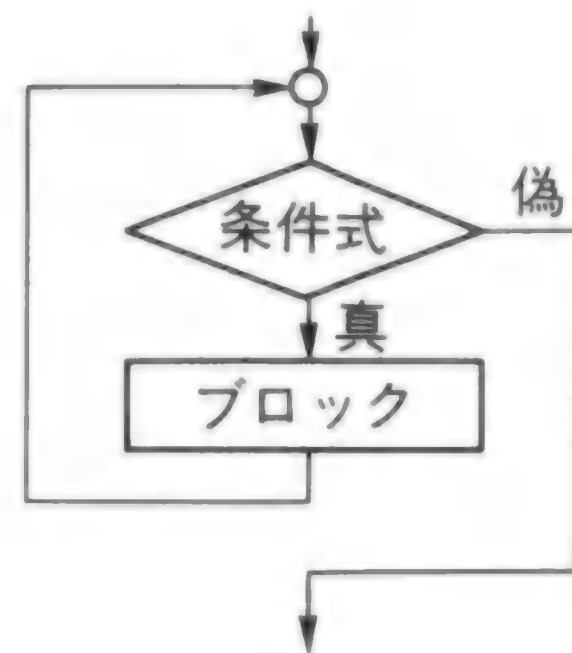
ブロック

WEND

DO WHILE 条件式

ブロック

LOOP

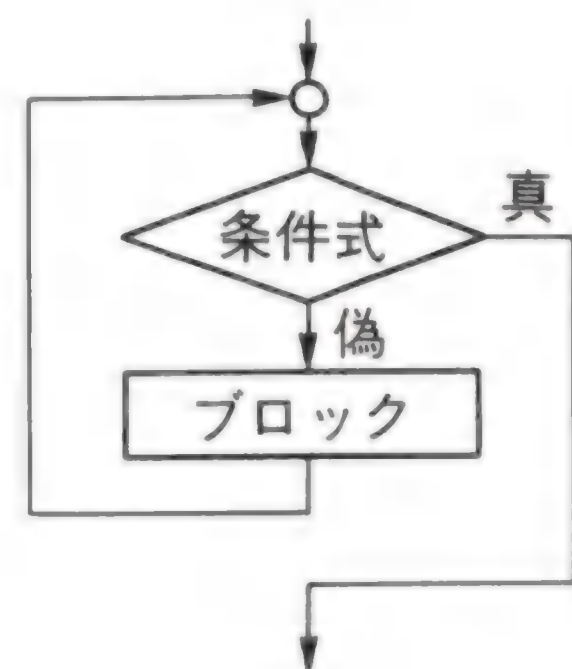


〈条件式〉が真の間，〈ブロック〉をくり返します。

DO UNTIL 条件式

ブロック

LOOP

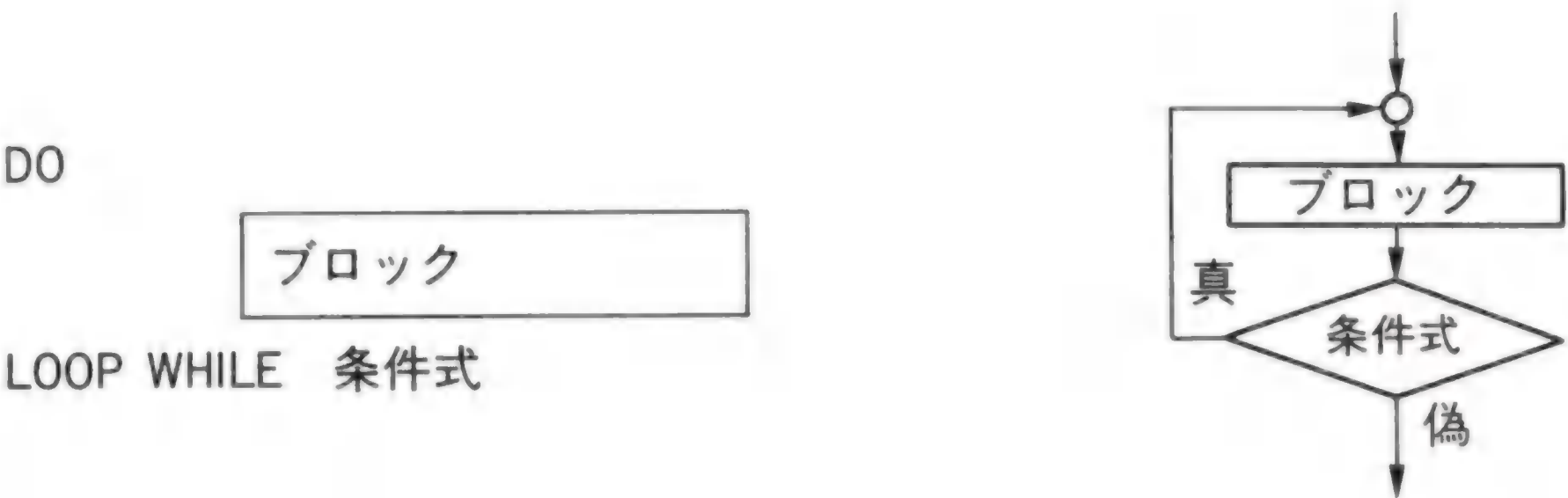


〈条件式〉が偽の間(真になるまで)，〈ブロック〉をくり返します。

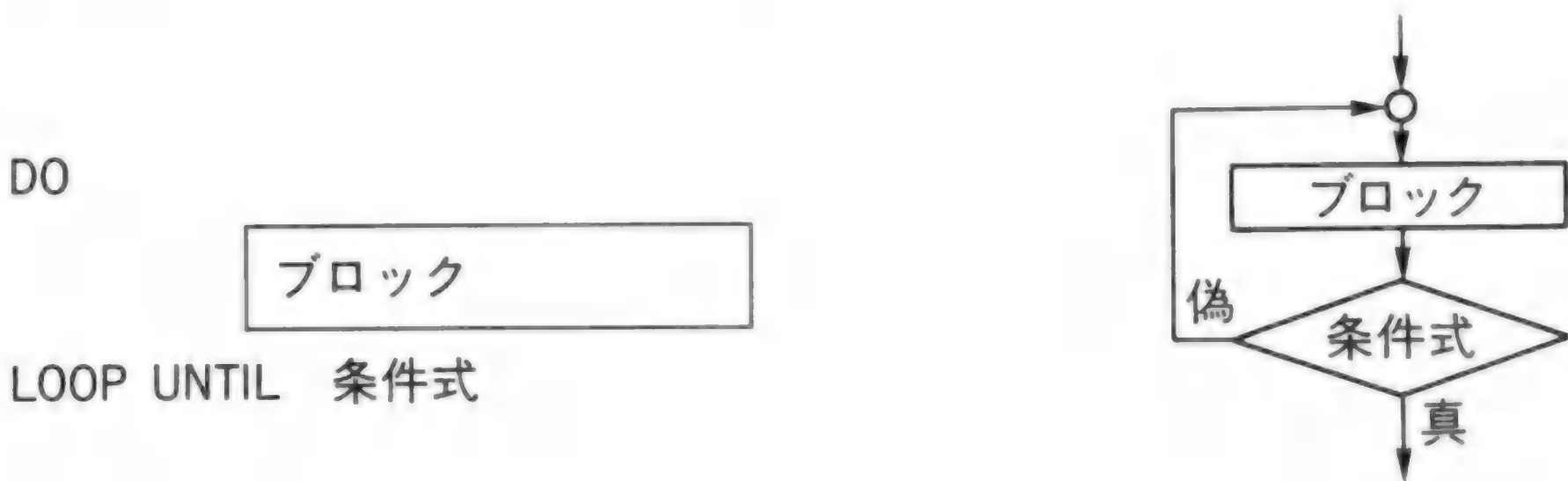
■後判定反復（DO~LOOP WHILE 文, DO~LOOP UNTIL 文）

後判定反復は、くり返し回数が定まっていないくり返しを行います。そして、くり返しの終了判定をループの後部で行います。

後判定反復を行う制御文として、DO~LOOP WHILE 文と DO~LOOP UNTIL 文があります。



〈条件式〉が真の間、〈ブロック〉をくり返します。



〈条件式〉が偽の間(真になるまで)、〈ブロック〉をくり返します。

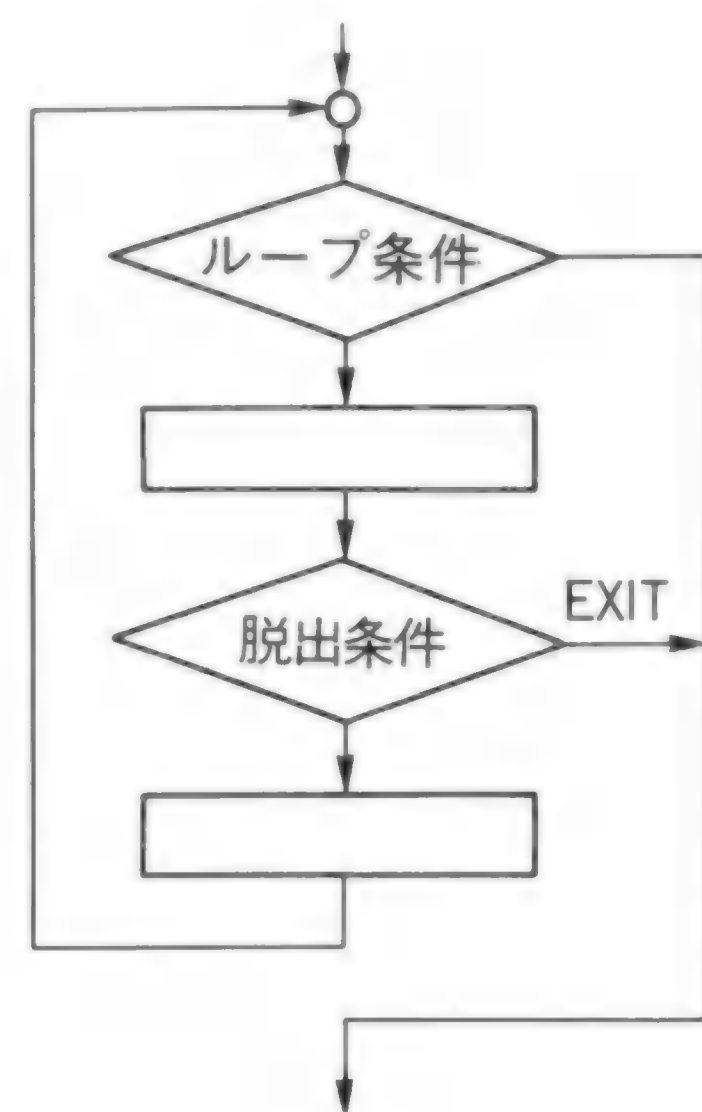
4 とび越し

ほかの場所にとび越しを行う制御文として、EXIT 文と GOTO 文があります。EXIT 文はループ中から強制脱出する際によく使いますが、GOTO 文はほとんど使う必要はありません。

■ EXIT 文

EXIT 文は、ループの途中でループから抜ける場合に使います。

```
DO WHILE 条件式
  ⋮
  IF 脱出条件 THEN EXIT DO
  ⋮
LOOP
```



EXIT DO は DO～LOOP から抜けるものですが、この他に EXIT DEF, EXIT FOR, EXIT FUNCTION, EXIT SUB があります。

■ GOTO 文

```
GOTO OWARI
⋮
OWARI: ←
```

構造化プログラミングという考え方では、GOTO 文はプログラムの流れをわかりにくくするのでできるだけ使わないようにしています。

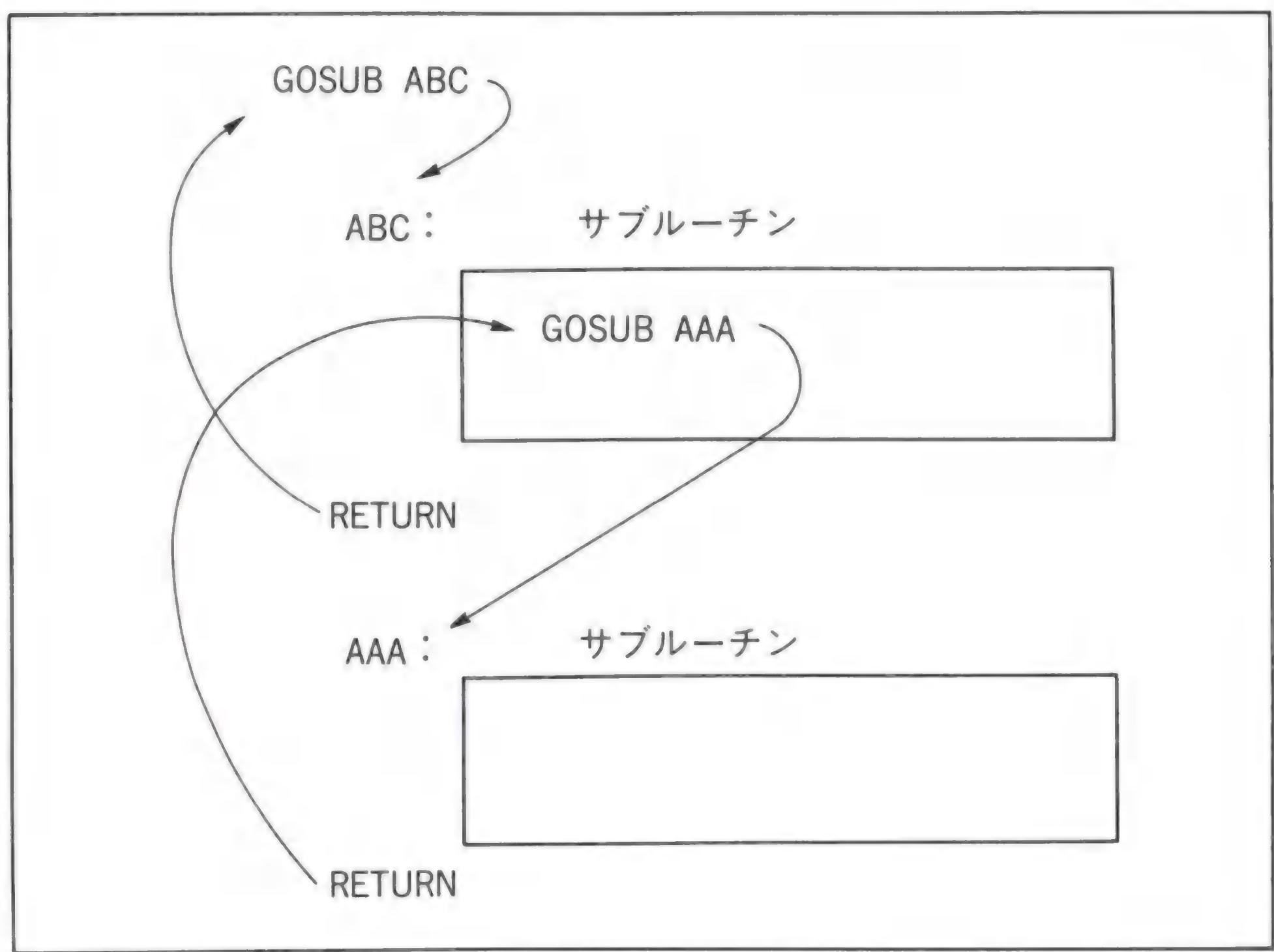
4-5 プロシージャ

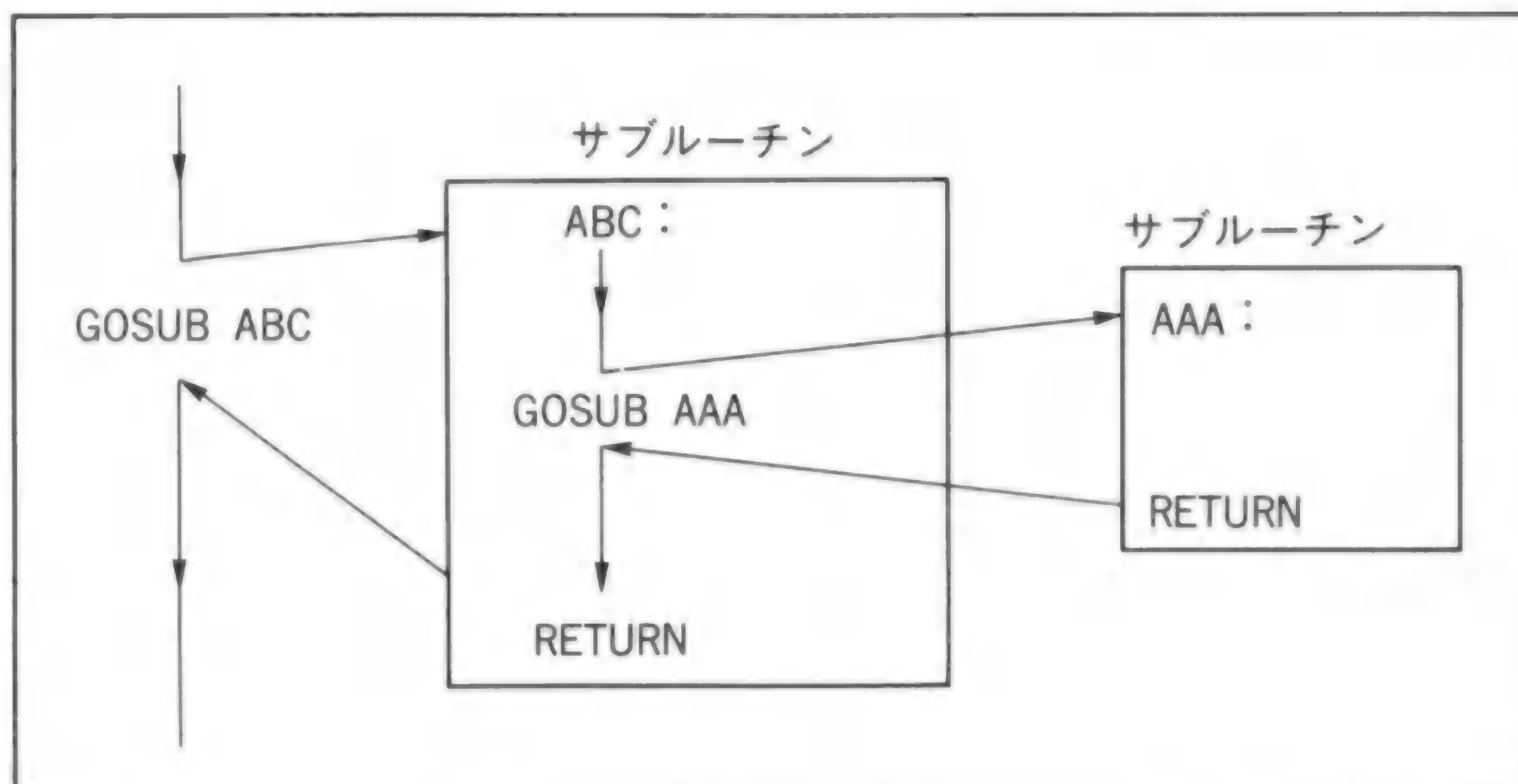
1 従来型サブルーチンとプロシージャ

従来型のサブルーチン(GOSUB 文で呼び出すもの)はメインルーチンに従属する一部として構成されます。

サブルーチンは、あるひとかたまりの処理を記述した小プログラムと考えることができます。サブルーチンはGOSUB 文により必要に応じて呼び出すことができます。サブルーチンの入口は行番号またはラベルで管理されます。サブルーチンのRETURN 文が実行されると、GOSUB 文の次の文に戻ってきます。サブルーチンから別のサブルーチンを呼び出すこともできます。

このプログラムが実行されたときの流れを下図に示します。





従来型の BASIC で扱っているサブルーチンは、次のような欠点を持っていました。

- ・ サブルーチン間のデータ授受は共通変数を介して行うため、それと異なる変数をサブルーチンに渡す場合は煩雑になる。
- ・ メインルーチンとサブルーチンで使う変数はすべて共通なため、各ルーチン間の独立性が保てない。

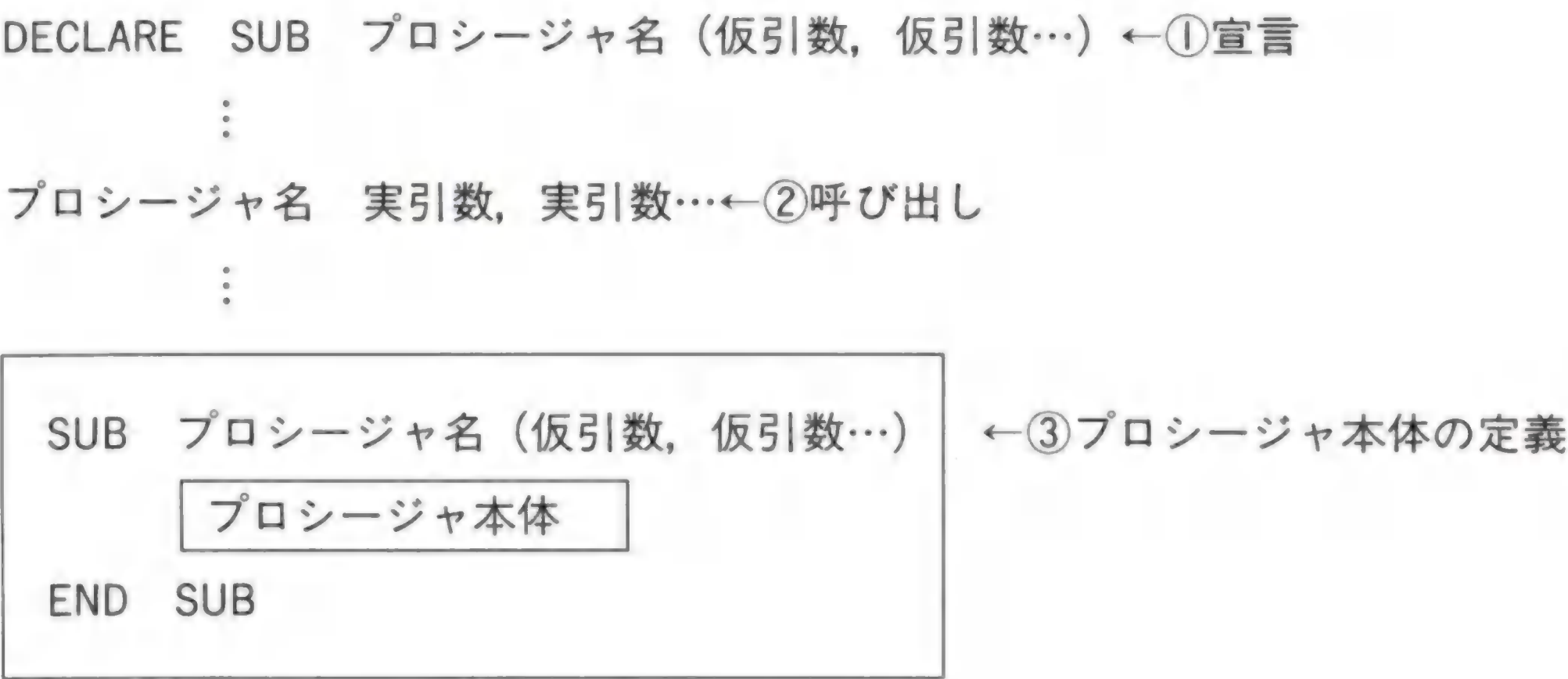
Quick BASIC では、従来型 BASIC の欠点を改良したサブルーチンをサポートします。この新しい形態のサブルーチンをプロシージャ(手続き)と呼びます。プロシージャは次の特徴を持ちます。

- ・ プロシージャ間のデータ授受は引数(ひきすう)を用いて行うため、汎用的なデータ授受が行える。
- ・ 各プロシージャ内の変数はそれぞれ独立(これをローカル変数という)なので、同じ名前の変数を使っても混同されない。
- ・ ほかのモジュールから使用することができる。

Quick BASIC のプロシージャは、SUB~END SUB(サブルーチン・プロシージャ)、FUNCTION~END FUNCTION(関数プロシージャ)の2種類で実現されます。

2 サブルーチン・プロシージャ

サブルーチン・プロシージャの定義方法，呼び出し方法を以下に示します。



①プロシージャの宣言

メインルーチンにおいて，プロシージャの呼び出しを可能とするために，プロシージャの名前と引数を宣言します。

もし，この DECLARE 文をプログラマが記述しなければ，プログラムをセーブしたときに QB のエディタがプロシージャの定義部を参照して自動的に生成します。

②プロシージャの呼び出し

DECLARE でプロシージャを宣言しているので，プロシージャ名を書くだけで (CALL 命令を使わずに) プロシージャを呼び出すことができます。プロシージャ名の後に，プロシージャに渡すデータをカンマ(,)で区切って書きます。これを実引数(じつひきすう)といいます。

③サブルーチン・プロシージャの定義

サブルーチン・プロシージャは SUB と END SUB のあいだに定義します。SUB の後にプロシージャ名を書き，その後にデータを受けとる変数名をカンマ(,)で区切って書きます。これを仮引数(かりひきすう)といいます。

仮引数には型宣言をすることもできますが，省略すると暗黙の型宣言とみなされます。

3 関数プロシージャ

サブルーチン・プロシージャと関数プロシージャの大きな違いは、前者が戻り値を持たないのに対し、後者は戻り値を持つことです。

関数プロシージャの定義方法、呼び出し方法を以下に示します。

DECLARE FUNCTION プロシージャ名 (仮引数, 仮引数…) ←①宣言

⋮

A = プロシージャ名 (実引数, 実引数, 実引数…) ←②呼び出し

⋮

FUNCTION プロシージャ名 (仮引数, 仮引数…) ←③プロシージャ本体の定義

プロシージャ名 = 値

END FUNCTION

①関数プロシージャの宣言

メインルーチンにおいてプロシージャの呼び出しを可能とするために、プロシージャの名前と引数を宣言します。

なお、この DECLARE 文をもしプログラマが記述しなければ、プログラムをセーブしたときに QB のエディタがプロシージャの定義部を参照して自動的に生成します。

②関数プロシージャの呼び出し

関数プロシージャの呼び出しは、「関数プロシージャ名(引数, …)」で行います。関数プロシージャの呼び出しは単独の文ではなく、式の中の一部として記述します。SIN(), COS()などの関数の扱いと同じです。

③関数プロシージャの定義

関数プロシージャの定義は、FUNCTION と END FUNCTION の間にします。FUNCTION の後に関数プロシージャ名を書き、その後にデータを受け取る引数をカンマ(,)で区切って書きます。

仮引数には型宣言をすることもできますが、省略すると暗黙の型宣言とみなされます。

関数プロシージャは値を呼び出し元に返すため、必ず一度、

関数プロシージャ名=□

という代入文を行わなければなりません。この□の値が、関数プロシージャの値(戻り値)として呼び出し元に返されます。

■ サブルーチン・プロシージャと関数プロシージャの違い

サブルーチン・プロシージャと関数プロシージャの違いを以下の表にまとめました。この点以外は、サブルーチン・プロシージャも関数プロシージャもまったく同じです。

● サブルーチン・プロシージャと関数プロシージャの違い

	サブルーチン・プロシージャ	関数プロシージャ
呼び出し方	1つの文として行う 実引数は()で囲まない Disp "Hello", 5, 3	式の中の一部として行う 実引数は()で囲む Heikin = Sum(a(), 10) / 10
戻り値	サブルーチン・プロシージャ自体は値を持たない	関数プロシージャ自体が値を持つ。これを戻り値という。つまり Sum(a(), 10) が合計の結果の値を持っていると考えられる

4 プロシージャの留意事項

QBでは各プロシージャをそれぞれ別のウィンドウで管理しているため、メインルーチンとプロシージャを同じウィンドウ内では見られません。各プロシージャおよびメインルーチンは、**[SHIFT]+[F・2]**により順次切り換えて画面に表示できます。

また、QBのエディタ上で「SUB プロシージャ名(引数) ☒」と入力すると、自動的にプロシージャを記述するウィンドウに切り換わります。

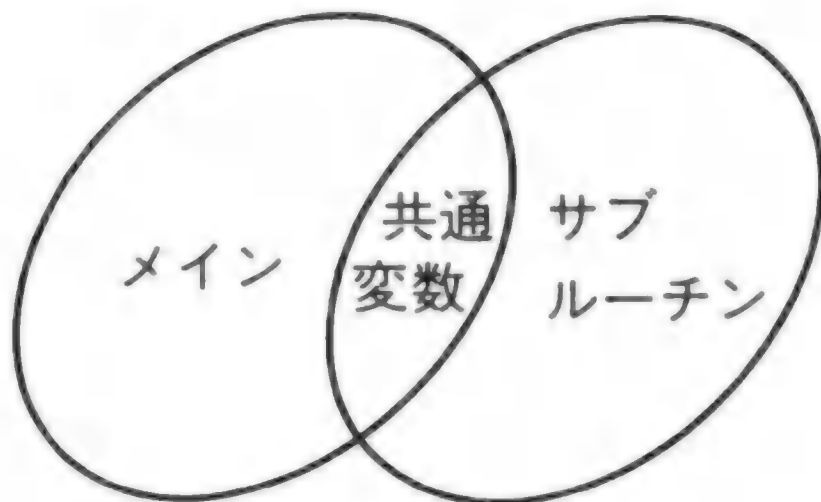
プロシージャのQB上での操作方法は第2章2-4の5を参照してください。

プロシージャの呼び出し時に書く引数は、プロシージャに渡す実際のデータなので実引数(じつひきすう)と呼び、変数、定数、式が書けます。

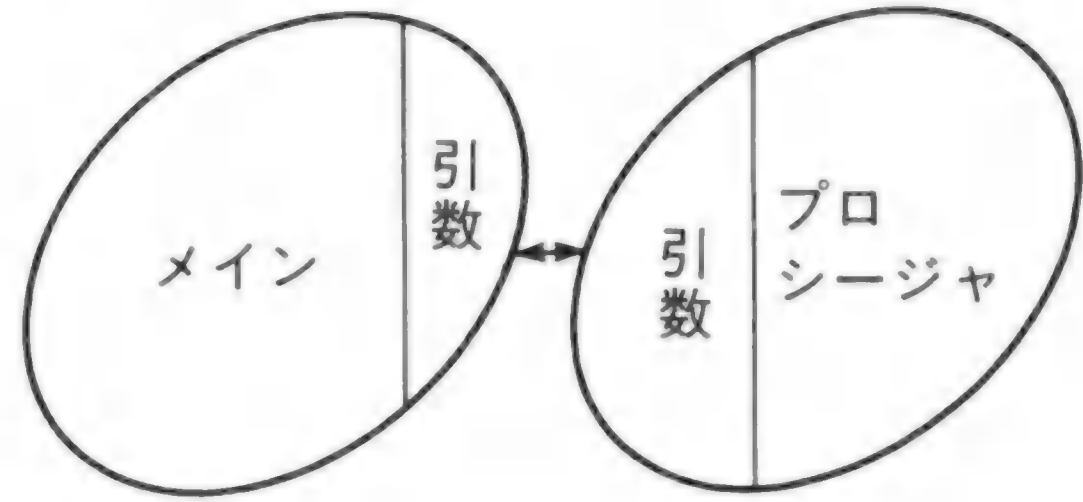
これに対し、プロシージャの定義側で書く引数を仮引数(かりひきすう)と呼びます。仮引数はデータを受け取るものなので変数しか指定できません。

実引数と仮引数の名前は違っていてもよいですが、データの型は一致していなければなりません。

●従来型サブルーチン



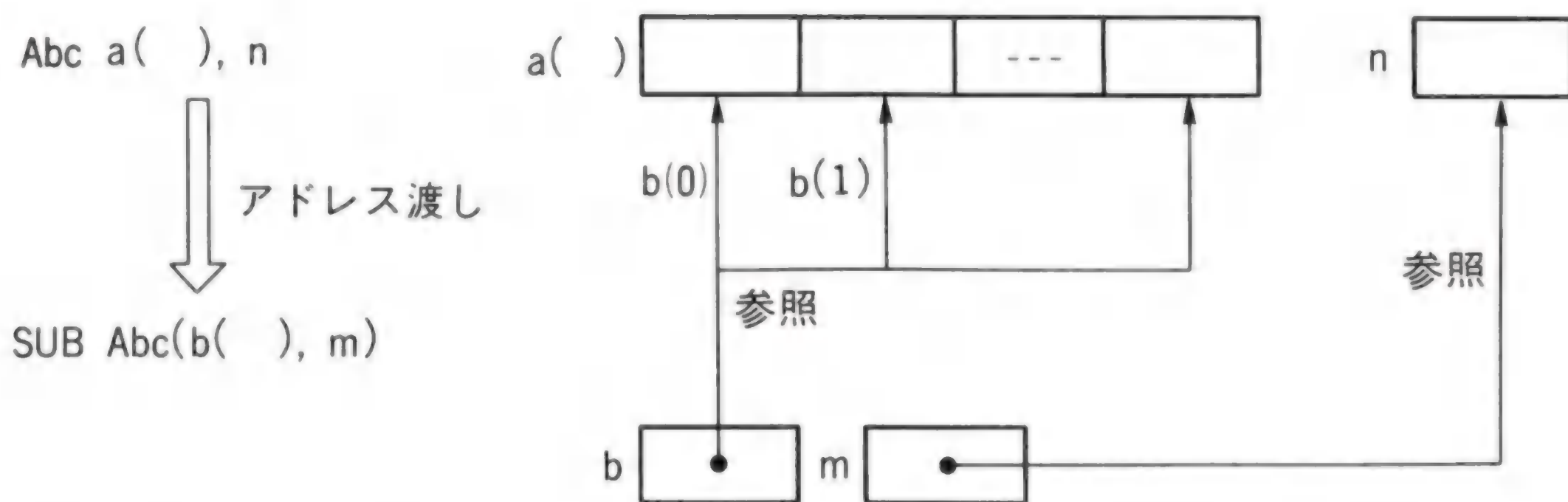
●プロシージャ



Quick BASIC の引数渡しでは、参照による呼び出し(call by reference)と、値による呼び出し(call by value)という2つの方法が行えます。

■参照による呼び出し (call by reference)

Quick BASIC のプロシージャ間の引数渡しは、原則的には参照による呼び出しで行われます。



参照による呼び出しでは、実引数のアドレスが仮引数に渡され、プロシージャ側ではこのアドレスをもとに、呼び出し元の仮引数を参照します。したがって、プロシージャ側で引数の値を変更すると、呼び出し側の引数の値も変化することになります。

つまり、仮引数は実引数と連動していて、プロシージャ側で仮引数の値を変えればその結果は実引数にも影響をおよぼします。


```

a$ = "Quick"
n = 3
Disp a$, n, 5
:
SUB Disp (Moji$, Num, Col)
    Moji$ = "Turbo"
    Num = 7
END SUB

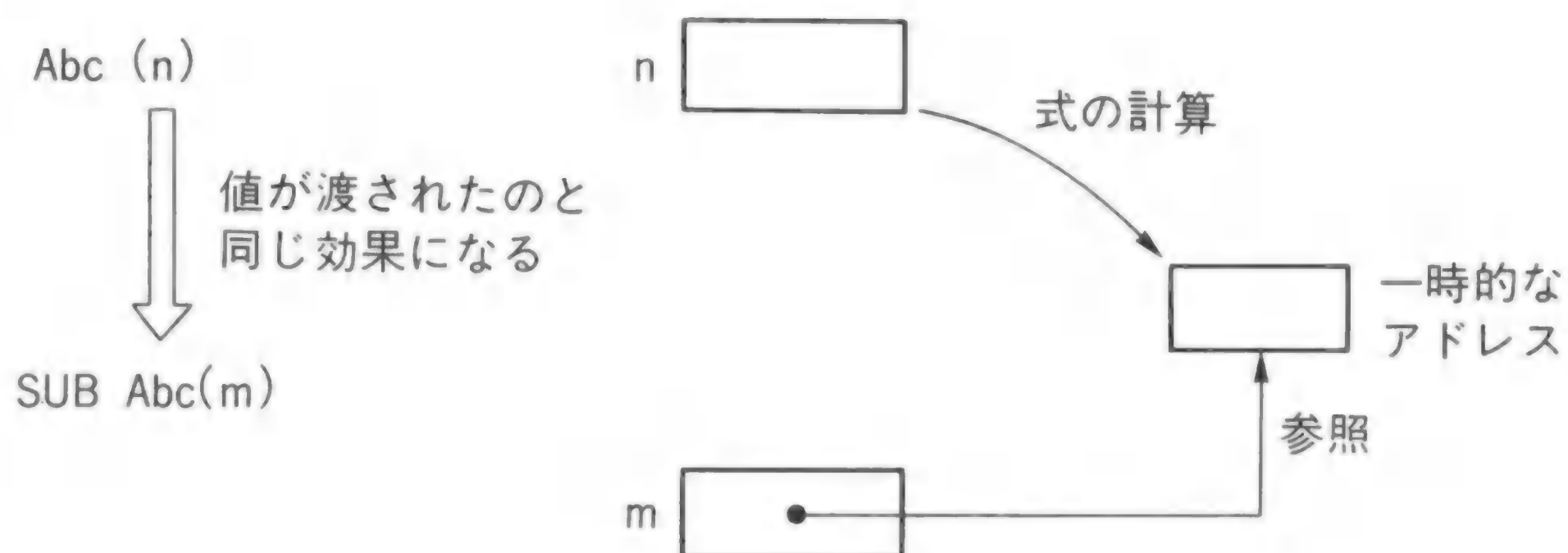
```

プロシージャで引数 Moji\$, Num の値を変えていますが、これは同時にメインルーチンの引数 A\$, n の値を変えていることになるのです。したがって、プロシージャから戻ったときの a\$, n の値はそれぞれ“Turbo”, 7 になっています。

これはあたかも、Moji\$, Num を介してメイン側の a\$, n に値を返しているようなものです。

■値による呼び出し (call by value)

Quick BASIC の引数渡しは、あくまでも参照による呼び出しで行っていますが、仮引数を式として与えることにより、値による呼び出しと同じ機能を実現できます。式とは $n + 1$, 5, (n) などです。() をつけば、式にできることに注意してください。

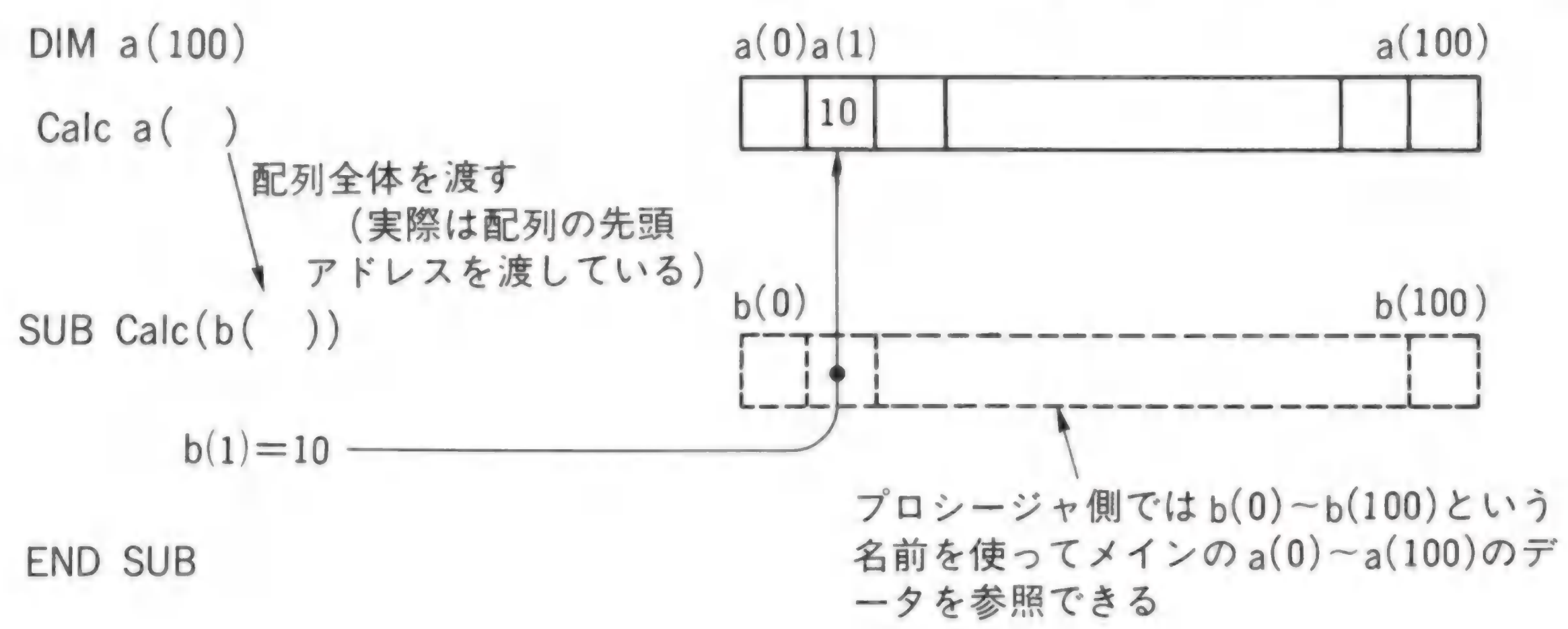


仮引数を式とすることで、その値が一時的なアドレスに記憶され、そのアドレスが実引数に渡されることになります。したがって、プロシージャ側で m の値を変えても、呼び出し側の n の値は変化を受けません。

値による呼び出しは、再帰プロシージャを記述する際に使います。

■配列データを渡す

配列データの全体をプロシージャに渡すには、引数として配列名の後に添字のない () を付けた「配列名 ()」を用います。



プロシージャ側では、b(0)～b(100)という配列を使ってメイン側のa(0)～a(100)という配列を操作することができます。また、プロシージャ側でb(1)=10などとすると、メイン側のa(1)も10に変わります。

このように、メイン側のa(0)～a(100)とプロシージャ側のb(0)～b(100)は連動していて、プロシージャ側でb(0)～b(100)を変化させると、メイン側のa(0)～a(100)も変化します。このことを利用して、プロシージャ側からメイン側に配列データを返すことができます。

なお、プロシージャ側では、渡された配列について、その先頭アドレスしかわかっていません。配列のサイズを調べたいときは、LBOUND / UBOUND関数を用いてください。

■レコード・データを渡す

レコード・データは、全体でもメンバ単位でもプロシージャに渡すことができます。

```
TYPE Eisei
    Star AS STRING * 10
    Kodo AS INTEGER
    Shuki AS SINGLE
END TYPE

DIM a AS Eisei
:
Disp a
:
SUB Disp (b AS Eisei)
:
END SUB
```

レコード・データ全体を渡す

仮引数の型宣言をする

6 DEF FN 関数

DEF FN 関数は、関数プロシージャと似ていますが、プロシージャではありません。DEF FN 関数が定義されているモジュール以外から呼び出すことはできません。DEF FN 関数は次のように定義します。

```
DEF FNMax(a,b)
  ⋮
  FNMax=□
END DEF
```

関数名はFNで始めます。DEF FN 関数の引数渡しは、値による呼び出し(call by value)で行いますから、関数側から呼び出し元に引数を介してデータを返すことはできません。

DEF FN 関数は再帰的に使うことはできません。

7 再帰

Quick BASIC では、再帰呼び出し(recursive call)が行えます。再帰呼び出しは、プロシージャの内部から自分自身のプロシージャを呼び出すコール方法です。再帰呼び出しは、一般に次のような形式をとります。

```
  ⋮
rproc [実引数, ...] ← 再帰プロシージャの最初の呼び出し
  ⋮
SUB rproc ([仮引数, ...]) ← 再帰プロシージャの定義
  IF 脱出条件 THEN EXIT SUB ← 再帰呼び出しからの脱出

  rproc [実引数, ...] ← 再帰呼び出し

END SUB
```


例

ハノイの塔問題の再帰解を QB でプログラムしたものです。

```
' ---/* ハノイの塔問題の再帰解 */---

DECLARE SUB Hanoi (n%, a$, b$, c$)

INPUT "円盤の枚数 "; n%
Hanoi n%, "a", "b", "c"
END

SUB Hanoi (n%, a$, b$, c$)
  IF (n% > 0) THEN
    Hanoi n% - 1, a$, c$, b$
    PRINT n%; "番の板を "; a$; " から "; b$: " に移動"
    Hanoi n% - 1, c$, b$, a$
  END IF
END SUB
```

円盤の枚数 ? 4

```
1 番の板を a から c に移動
2 番の板を a から b に移動
1 番の板を c から b に移動
3 番の板を a から c に移動
1 番の板を b から a に移動
2 番の板を b から c に移動
1 番の板を a から c に移動
4 番の板を a から b に移動
1 番の板を c から b に移動
2 番の板を c から a に移動
1 番の板を b から a に移動
3 番の板を c から b に移動
1 番の板を a から c に移動
2 番の板を a から b に移動
1 番の板を c から b に移動
```

4-6

メタコマンド

1 メタコマンドとは

メタコマンドは、コメントステートメント('または REM)の後に次のように書きます。

'\$メタコマンド

メタコマンドは、プログラムの実行に先立って、Quick BASIC に対し、ある処理を指示するもので、\$INCLUDE、\$DYNAMIC、\$STATIC の 3 つがあります。

2 \$INCLUDE

\$INCLUDE は、別のプログラム・ファイルを取り込むもので次のように書きます。

'\$INCLUDE: 'ファイル名'

これで、この位置に<ファイル名>で指定されるファイルを取り込みます。

インクルード・ファイルはテキストファイルでなければなりません。インクルード・ファイルの作り方は、第 2 章 2-4 の 7 を参照してください。

インクルード・ファイルにはプロシージャを記述できません。

3 \$DYNAMIC, \$STATIC

\$DYNAMIC と \$STATIC は、配列の記憶領域の割り当て方を指示するもので、前者は動的配列、後者は静的配列の割り当てを指示します。

```
'$DYNAMIC
DIM a(100) ← 動的配列
'$STATIC
DIM b(100)
DIM c(100) } ← 静的配列
```


第5章

ステートメント と関数の概要

5-1

実行制御

プログラムの流れを制御するステートメントで、プログラムを作る上で基本になるものです。詳しくは第4章4-4を参照してください。

ステートメント	機能
CHAIN	別のプログラムに実行を移す
DO～LOOP	前判定/後判定反復
EXIT	ループ、プロシージャからの脱出
FOR TO STEP～NEXT	所定回反復
GO TO	分岐
GOSUB	サブルーチンの呼び出し
IF GOTO ELSE	単純 IF
IF THEN ELSE	単純 IF
IF THEN～ELSEIF～END IF	ブロック IF
ON GOSUB	条件によるサブルーチン・コール
ON GOTO	条件による分岐
RETURN	サブルーチンからのリターン
SELECT CASE	複数条件判断
STOP	実行の停止
WHILE～WEND	前判定反復

5-2

サブモジュール

プロシージャの宣言，定義，呼び出しを行うステートメントです．詳しくは第 4 章 4-5 を参照してください．

ステートメント	機 能
CALL CALLS DECLARE SUB DECLARE FUNCTION DEF FN DEF FN～END DEF FUNCTION～END FUNCTION SUB～END SUB	プロシージャの呼び出し 引数を far アドレスとして渡すプロシージャ・コール サブルーチン・プロシージャの宣言 関数プロシージャの宣言 関数の定義（1 行） 関数の定義（複数行） 関数プロシージャの定義 サブルーチン・プロシージャの定義

5-3

変数管理

変数の型宣言，レコード型の定義，変数のスコープ指定，変数の初期化，変数の割り当てられているアドレスの取得などを行うステートメントと関数です。詳しくは第4章4-2を参照してください。

	ステートメント/関数	機 能
宣言と定義	COMMON CONST DIM DEF INT/SNG/DBL /LNG/STR SHARED STATIC TYPE～END TYPE	共用変数の宣言 記号定数の定義 変数または配列の型宣言 変数のインプリシット宣言 プロシージャ間の変数の共有 静的変数の宣言 レコード型の定義
初期化	CLEAR ERASE REDIM	変数の初期化 配列変数の初期化と解放 動的配列の割り当てスペースの変更
変数のアドレス	SADD (関数) VARPTR (//) VARPTR\$ (//) VARSEG (//)	文字列が格納されているアドレスの取得 変数のアドレスのオフセット値の取得 変数のアドレスを文字列として取得 変数のアドレスのセグメント値の取得
添字	OPTION BASE LBOUND (関数) UBOUND (//) LEN (//)	配列添字の下限デフォルト値の設定 配列添字の下限値の取得 配列添字の上限値の取得 変数のサイズの取得

セグメント値とオフセット値については，本章5-13を参照してください。

5-4 ファイル処理

ファイルのオープン/クローズ，ファイルとのデータの入出力，ファイル情報の取得を行うステートメントと関数です。

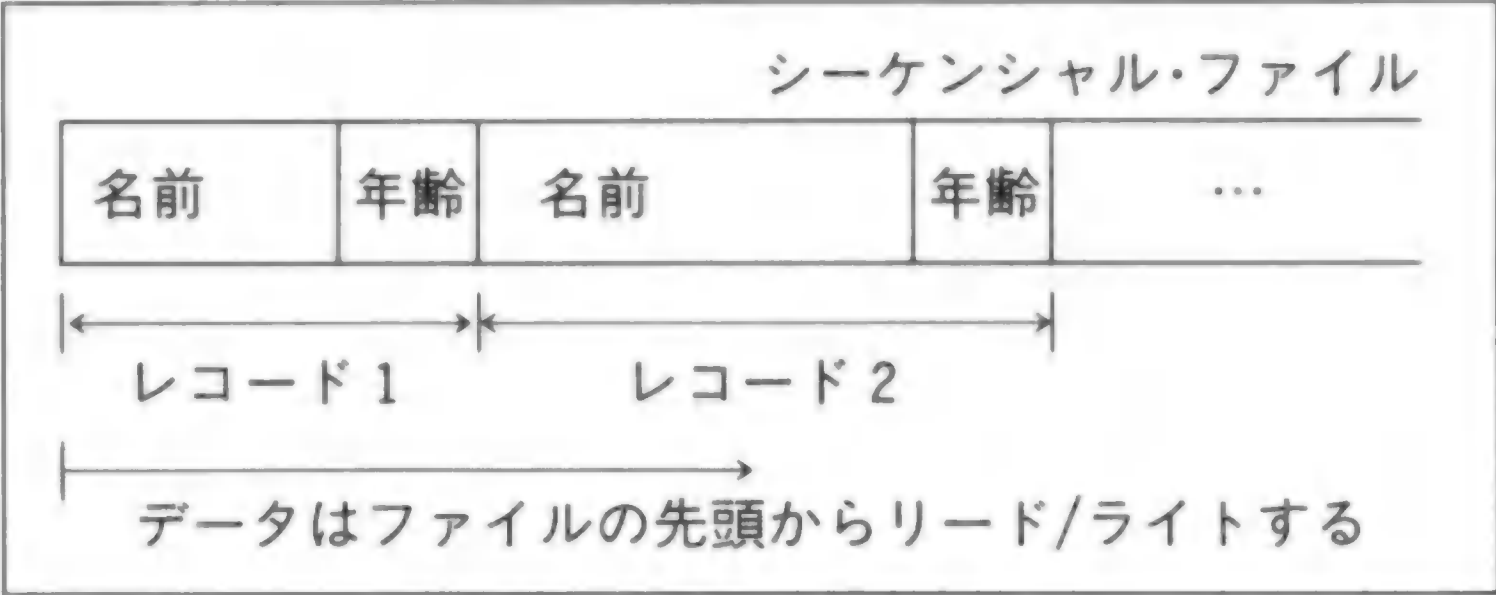
	ステートメント/関数	機 能
オープン	OPEN CLOSE LOCK UNLOCK	ファイルのオープン ファイルのクローズ 他のプロセスからのアクセスの禁止 他のプロセスからのアクセスの禁止の解除
シーケンシャル・アクセス	INPUT\$ (関数) INPUT¥ (//) INPUT # LINE INPUT # PRINT # WRITE #	指定したバイト数分の文字列のリード 指定した長さの文字のリード シーケンシャル・ファイルからのリード シーケンシャル・ファイルから1行単位でリード シーケンシャル・ファイルへのライト シーケンシャル・ファイルへのライト
ランダム・アクセス	FIELD LSET RSET GET # PUT #	ファイルバッファの設定 ファイルバッファへの値の設定 ファイルバッファへの値の設定 ランダム・ファイルからのリード ランダム・ファイルへのライト
ファイル情報	EOF (関数) FILEATTR (//) FREEFILE (//) LOC (//) LOF (//) SEEK (//) SEEK #	ファイル終わりの検知 ファイル属性の取得 未使用ファイル番号の取得 ファイル現在位置の取得 ファイルサイズの取得 ファイル現在位置の取得 ファイル現在位置の移動

■ファイルの種類

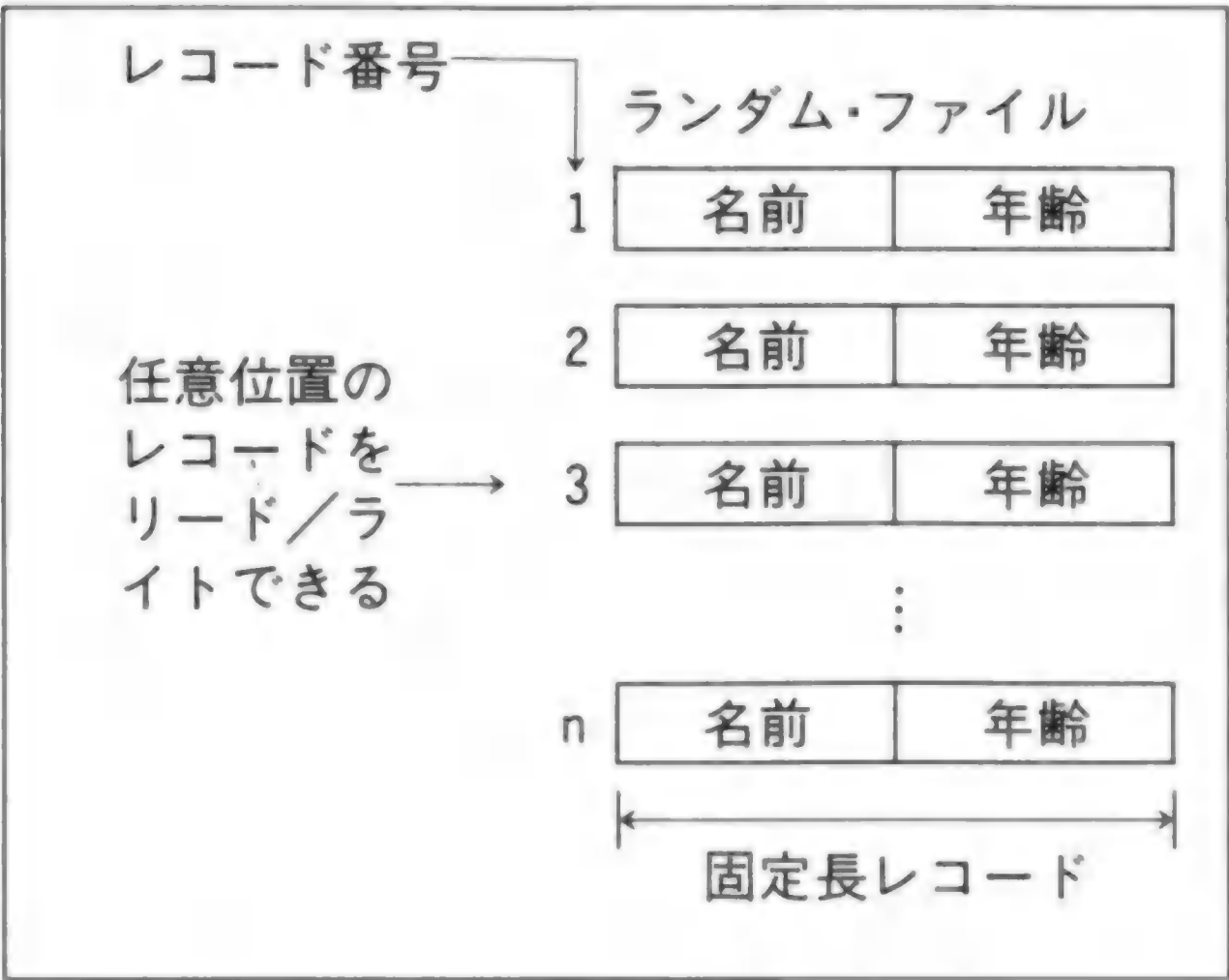
BASICで扱えるファイルは、シーケンシャル・ファイルとランダム・ファイルの2つです。

シーケンシャル・ファイルは、データをファイルの先頭から順次リード/ライトしていくものです。

1件のレコードの長さは可変長であってもかまいません。



これに対しランダム・ファイルは、任意のレコード位置のデータをリード/ライトすることができます。1件のレコードの長さは、固定長でなければなりません。



■ファイルのオープンとファイル番号

ファイルのオープンにより，ファイル名で示されるファイルを，指定したファイル番号(#n)で処理できるようになります。

```
OPEN  "d:filename.typ" FOR { INPUT
                             OUTPUT
                             APPEND } AS #n
                             ↑
                             ファイル番号
OPEN  "d:filename.typ" FOR  RANDOM AS #n  LEN= レコード長
```

オープンモードの意味は下表のとおりです。

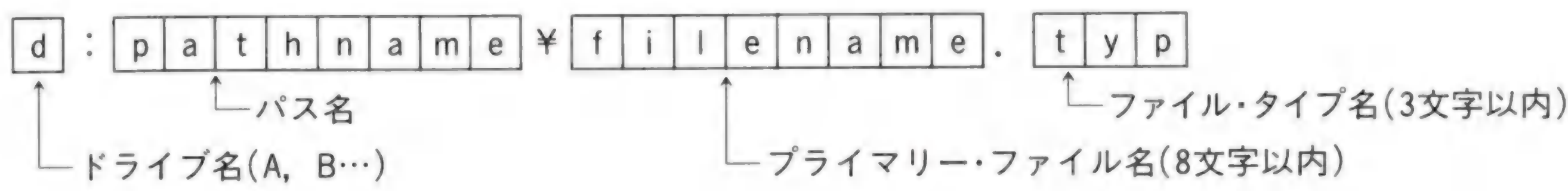
モード	意 味
INPUT OUTPUT APPEND	すでに作られているファイルからデータを読む 新規にファイルを作成し，データを書く すでにあるファイルの後にデータを追加書き込み
RANDOM	ランダム・ファイルとしてリード/ライトモードでオープン
BINARY	バイナリモードでオープン

ファイルへのデータ処理が終了したら，

```
CLOSE  #n
```

により，ファイルをクローズしてファイル番号を解放します。

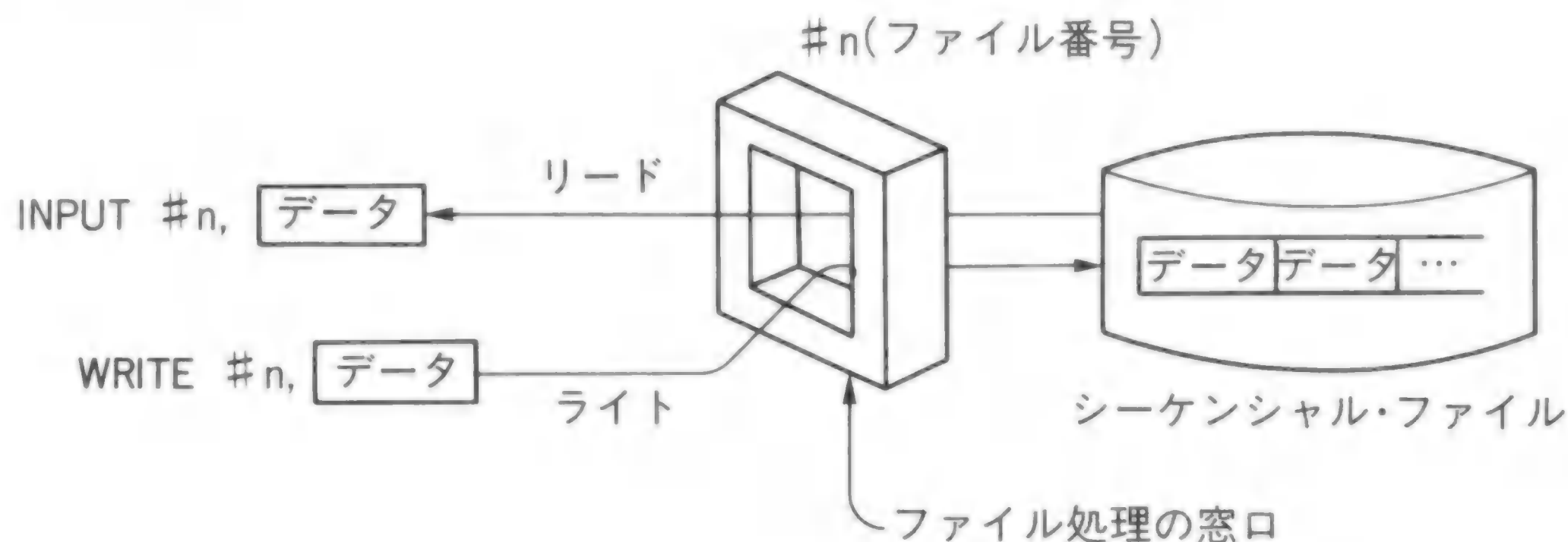
QBで扱うファイル名は，以下に示すMS-DOSのファイル名の規則にしたがいます。



なお，ファイル名の英字の大／小文字は区別されず同じものとして扱われます。つまり test.dat と TEST.DAT は同じファイル名となります。

■シーケンシャル・ファイルのリード/ライト

ファイル・オープンされているファイルには、ファイル番号を通してデータをリード/ライトします。

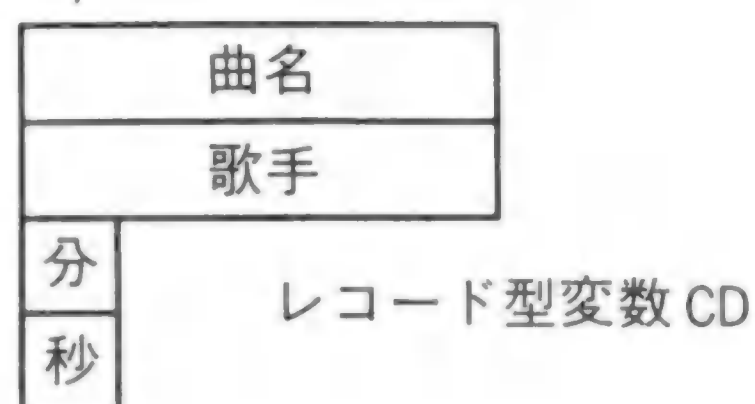
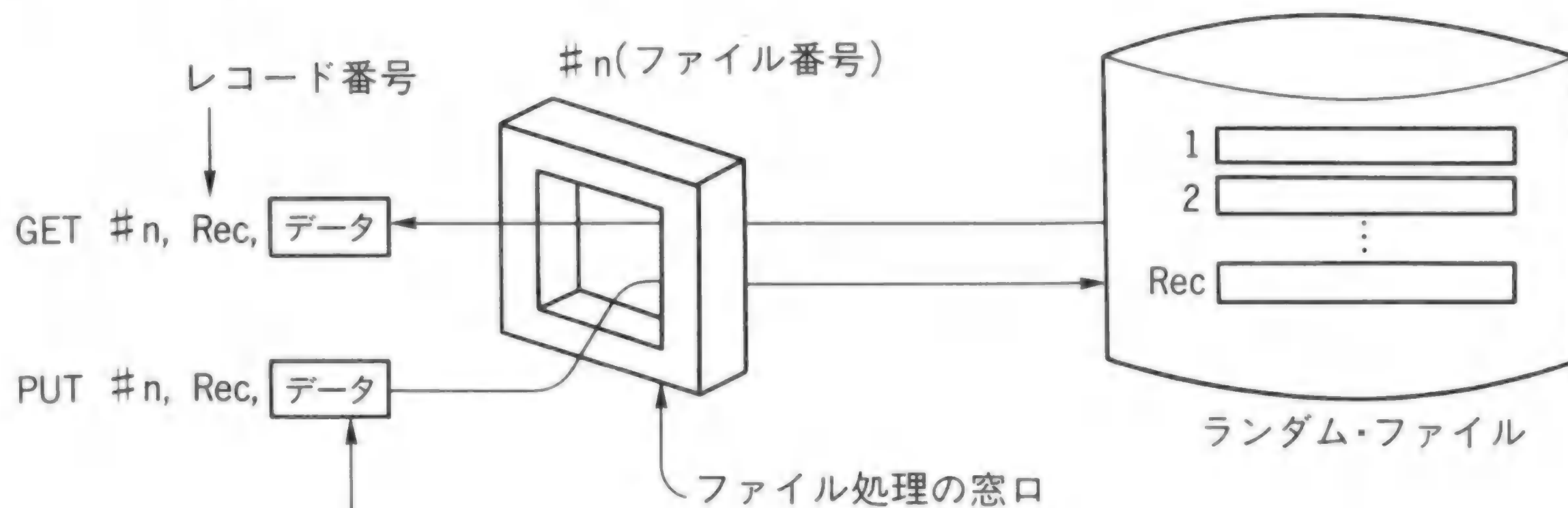


ファイルをリードしていったとき、ファイルのエンドはEOF関数で調べます。EOF(n)はファイル番号nのファイルがファイル・エンドなら真(-1)を返します。したがって、ファイル・エンドまでファイルからデータをリードするには、次のように書きます。

```
WHILE NOT EOF(n)
  INPUT #n, 
  ⋮
WEND
```

■ランダム・ファイルのリード/ライト

ランダム・ファイルは、指定したレコード番号のデータを即座にリード/ライトすることができます。



```
TYPE CDMusic
  Song AS STRING * 20
  Singer AS STRING * 20
  Min AS INTEGER
  Sec AS INTEGER
END TYPE

DIM CD AS CDMusic
```


レコードの長さは、

LEN(レコード型変数名)

で求められ、ファイルのサイズは、

LOF(ファイル番号)

で求められるので、ランダム・ファイルのレコード数は、

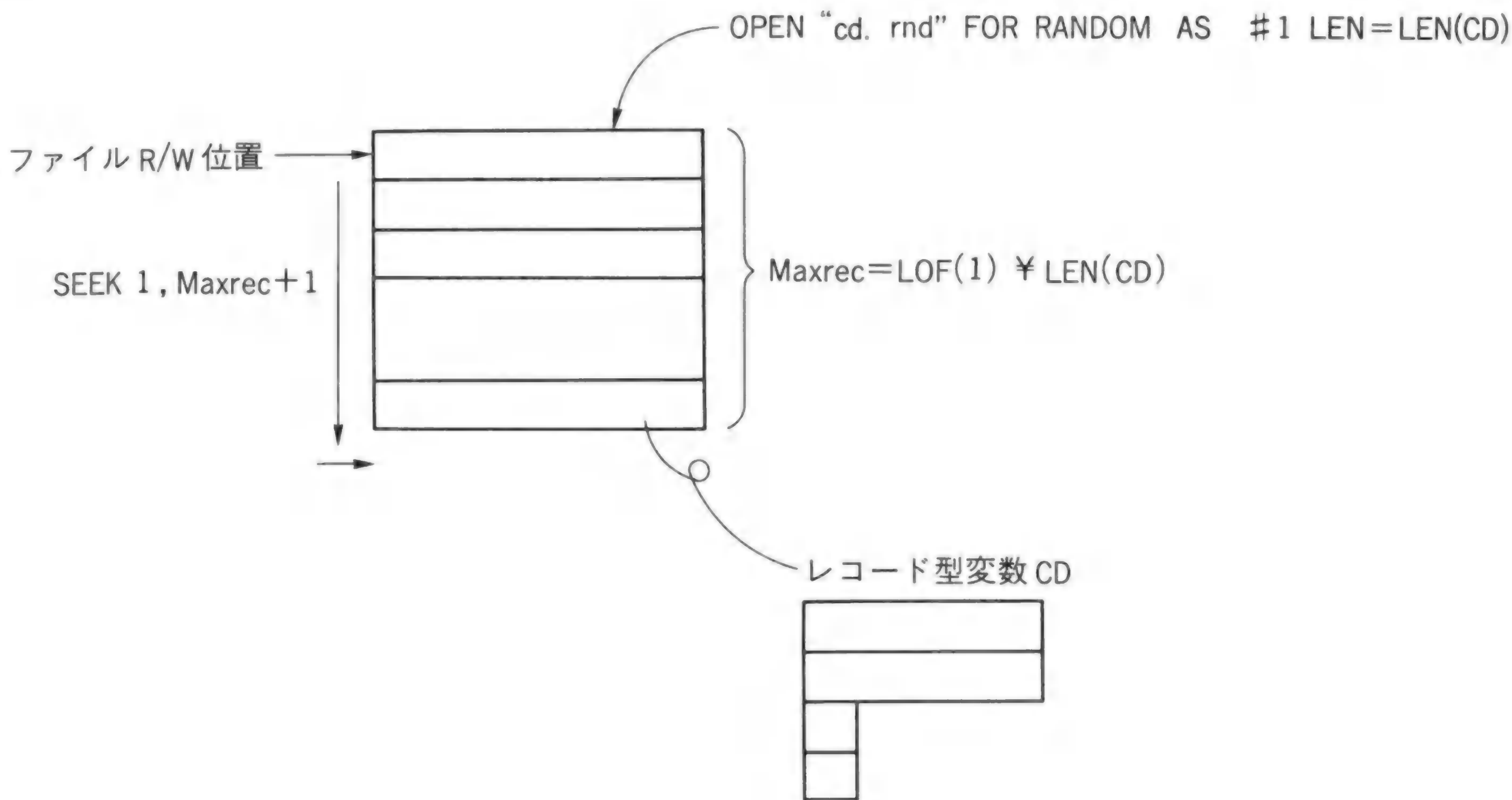
LOF(ファイル番号) ÷ LEN(レコード型変数)

で求められます。

したがって、ランダム・ファイル(ファイル番号 n)のファイル・エンドの次にリード/ライト位置を移動するには、

SEEK n, LOF(n) ÷ LEN(レコード型変数名)+1

とします。



■デバイス・ファイル

Quick BASIC では次のようなデバイスを OPEN によりオープンすることで、ディスク・ファイルと同様にファイルとして扱うことができます。

デバイス名	デバイス	入出力モード
COM1 :	シリアルポート1	入出力
COM2 :	シリアルポート2	入出力
CONS :	スクリーン	出力
KYBD :	キーボード	入力
LPT1 :	プリンタ1	出力
LPT2 :	プリンタ2	出力
LPT3 :	プリンタ3	出力
SCRN :	スクリーン	出力

たとえば、プリンタのオープンとそれへの出力は、次のように行います。

OPEN "LPT1:" FOR OUTPUT AS #2

⋮

PRINT #2, "HELLO"

←プリンタのオープン

←プリンタに"HELLO"を出力

5-5

コンソール入出力

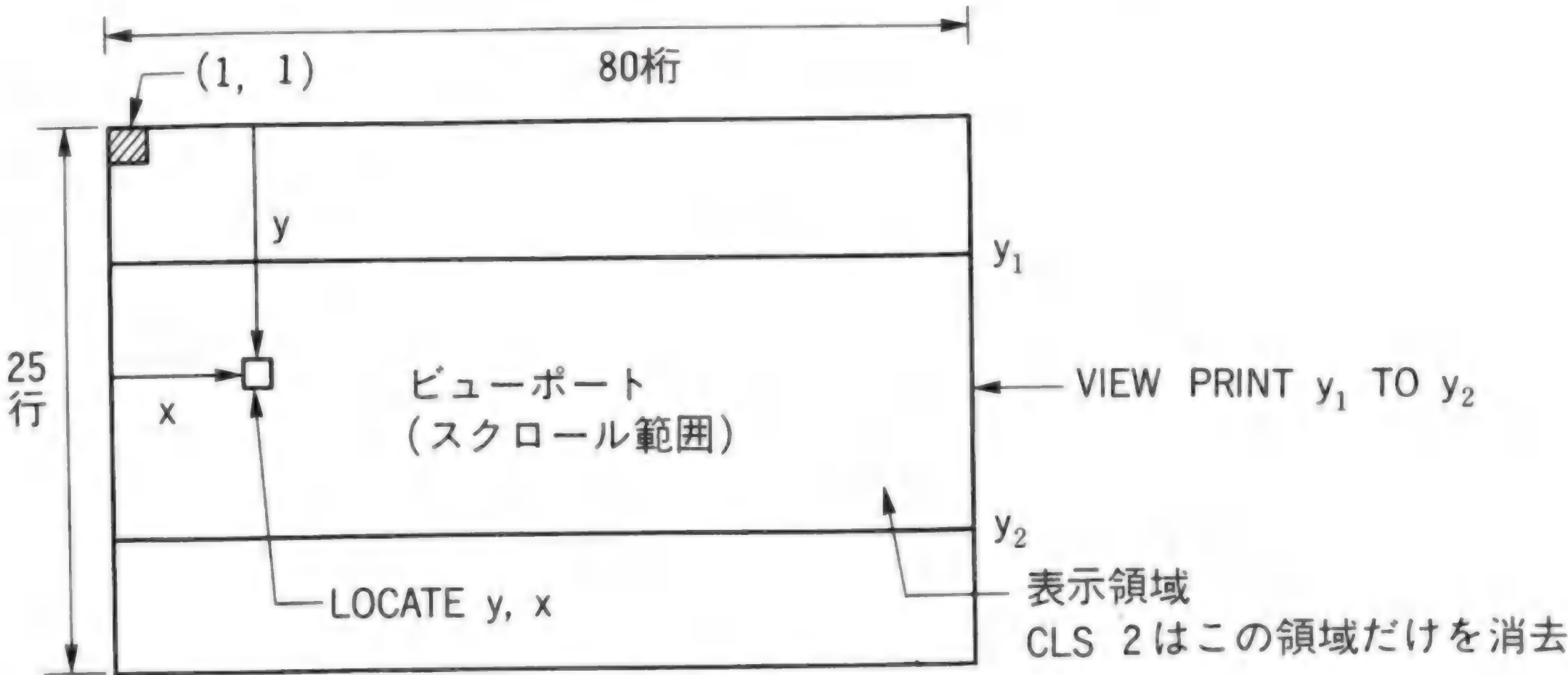
キーボードからの入力，スクリーン(CRT)への出力を行うステートメントと関数です。

	ステートメント/関数	機 能
キーボード	INKEY\$ INPUT KEY LINE INPUT	待ったなしのキー入力 キーボードからのデータ入力 ファンクションキー制御 キーボードから 1 行入力
スクリーン	CSRLIN (関数) LOCATE POS (関数) PRINT PRINT USING SCREEN (関数) SPC (関数) TAB (関数) VIEW PRINT WIDTH WRITE	現在のカーソル行の取得 カーソルを指定位置に移す 現在のカーソル欄の取得 スクリーンへの出力 書式指定付きのスクリーンへの出力 指定位置の文字または色の取得 スペースの出力 表示位置の指定 テキストビューポートの上下行の指定 画面幅の設定 スクリーンへの出力

■画面構成

テキスト出力画面は80桁×25行のサイズで，左隅上の座標が(1, 1)です。WIDTH 文がありますが，98用 Ver.4.2の画面サイズは80×25の固定です。Ver.4.5では，80×20もサポートしています。

VIEW PRINT 文により画面上への表示範囲(ビューポート)を設定することができます。CLS 2により消去される範囲はこのビューポートの中だけです。LOCATEで指示する行位置は，ビューポートの開始行ではなく，物理画面の最上行です。



5-6

ディスク・ファイル操作

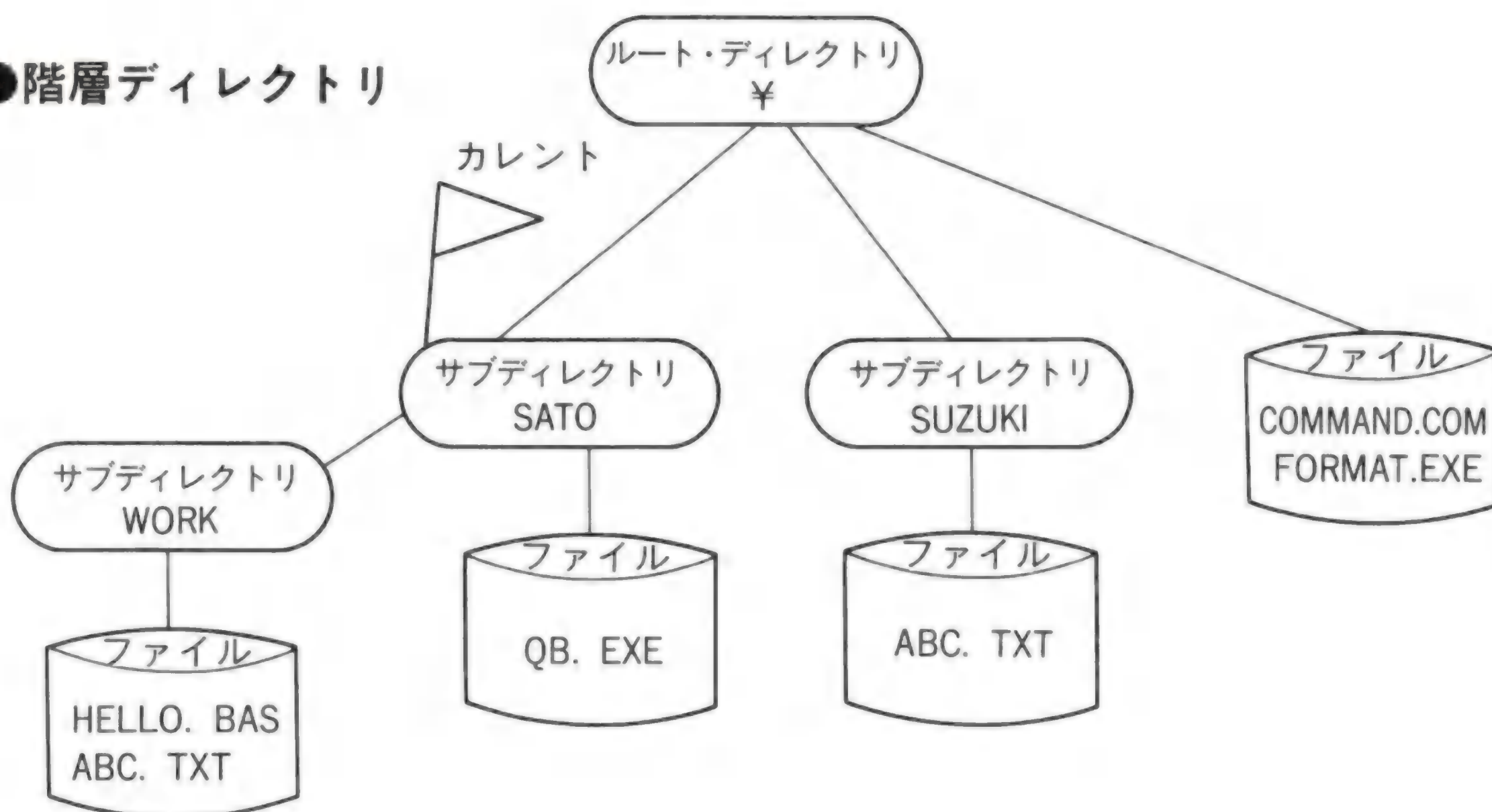
ディスク上のファイルと階層ディレクトリの操作を行うステートメントです。

	ステートメント	機 能
階層ディレクトリ	CHDIR MKDIR RMDIR	カレント・ディレクトリの変更 サブディレクトリの作成 サブディレクトリの削除
ファイル	FILES KILL NAME	ファイル一覧(ディレクトリ)の表示 ファイルの削除 ファイル名の変更

■階層ディレクトリ

1つのディスクを複数のユーザが共同利用する場合(特にハードディスク・システムでこの傾向が強い)や、ファイルの種類が増えてくると、単一のディレクトリ構造では効率的なファイル管理が行えません。そこで、図に示すような階層ディレクトリ構造が考えられています。

●階層ディレクトリ



階層ディレクトリでは、一番のもとになるディレクトリをルート・ディレクトリと呼び、その下にいくつかのサブディレクトリを持っています。現在の作業のもとになっているディレクトリをカレント・ディレクトリといいます。システム起動時はカレント・ディレクトリはルート・ディレクトリに設定されています。

5-7

デバイス操作

プリンタ，通信回線，入出力ポートなどに対する操作を行うステートメントと関数です。

	ステートメント/関数	機 能
プ リ ン タ	LPOS (関数) LPRINT WIDTH LPRINT	印刷行バッファ内のプリンタ・ヘッドの現在位置の取得 プリンタへのデータ出力 プリンタの 1 行幅の設定
ポ ー ト 入 出 力	OPEN COM (関数) INP OUT WAIT	通信ポートのオープン ポートから 1 バイト入力 ポートへ 1 バイト出力 入力ポートの状態を調べる間プログラムを停止
デ バ イ ス バ	IOCTL\$ (関数) IOCTL	デバイス・ドライバから制御文字列を取得 制御文字列をデバイス・ドライバに送る

5-8

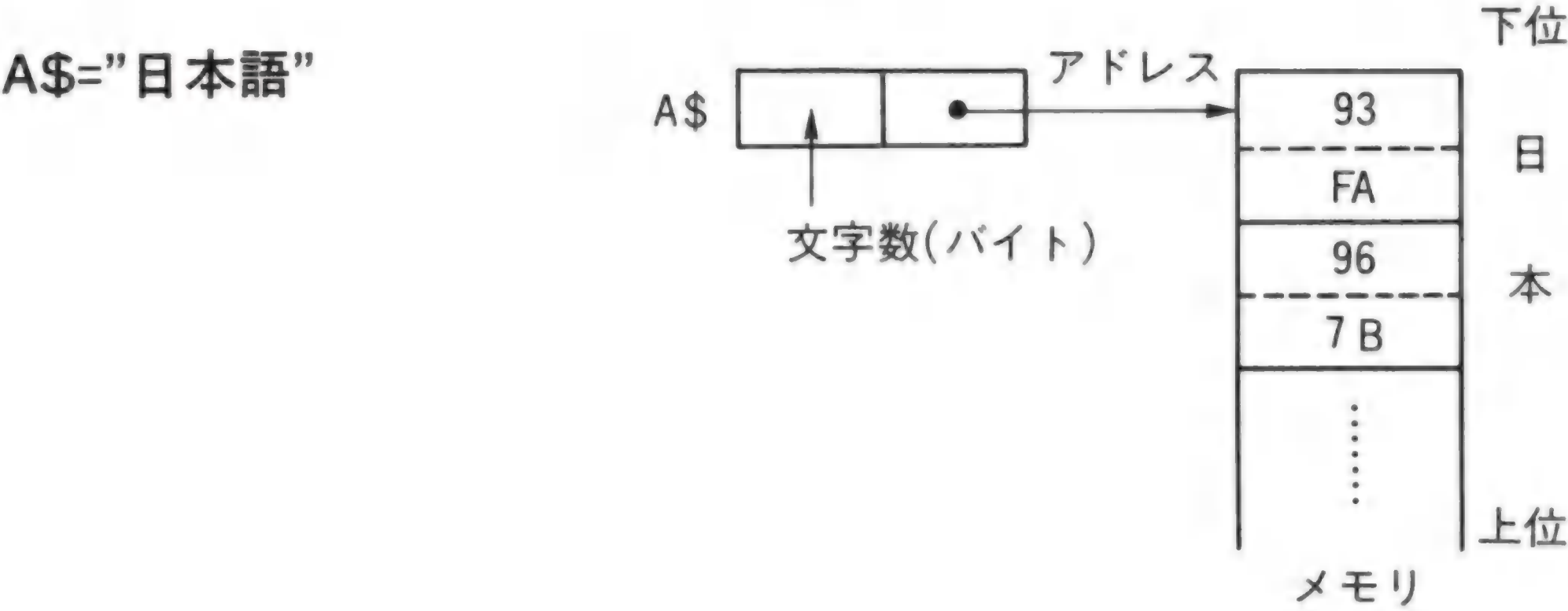
文字列処理

文字列に対する処理を行うためのステートメントと関数です。

	ステートメント/関数		機 能
一般文字列	INSTR		文字列の検索
	LEFT\$	(関数)	左部分文字列の取得
	LEN	(//)	文字列長さの取得
	MID\$	(//)	中間文字列の取得
	MID\$		文字列の置き換え
	RIGHT\$	(関数)	右部分文字列の取得
漢字	KINSTR		INSTR の漢字対応
	KLEN	(関数)	LEN の漢字対応
	KMID\$	(//)	中間文字列の取得
	KMID		文字列の置き換え
コード	ASC	(関数)	文字を ASCII コードに変換
	CHR\$	(//)	ASCII コードを文字に変換
	JIS\$	(//)	文字を JIS コードに変換
	KTN\$	(//)	文字を句点コードまたは ASCII コードに変換
変換	CDBL\$	(関数)	1 バイト文字を 2 バイト文字に変換
	CSNG\$	(//)	2 バイト文字を 1 バイト文字に変換
	LCASE\$	(//)	小文字に変換
	UCASE\$	(//)	大文字に変換
その他	KEXT\$	(関数)	1 バイト文字または 2 バイト文字の抽出
	LTRIM\$	(//)	左側スペースの削除
	RTRIM\$	(//)	右側スペースの削除
	SPACE\$	(//)	スペースの生成
	STRING\$	(//)	文字列の生成

■漢字の扱い

文字列中の漢字は、シフト JIS コードでメモリ上に上位バイト，下位バイトの順に格納されています。たとえば，次のようにです。



ところで，ASC,CHR\$で扱う漢字のコードはシフト JIS コードではなく，漢字連続コードという特殊なもので，次のように対応しています。

漢字連続コード	シフト JIS コード	漢字
256	&H8140	スペース
257	&H8141	、
⋮	⋮	⋮
⋮	⋮	⋮
1666	&H889F	亜
⋮	⋮	⋮

5-9

数値処理

数値処理を行うためのステートメントと関数です。

	ステートメント/関数		機 能
算術関数	ABS	(関数)	絶対値
	ATN	(//)	アークタンジェント
	COS	(//)	コサイン
	EXP	(//)	指数
	FIX	(//)	小数部切り捨て
	INT	(//)	小数部切り捨て
	LOG	(//)	自然対数
	SGN	(//)	正・負符号
	SIN	(//)	サイン
	SQR	(//)	ルート
	TAN	(//)	タンジェント
乱数	RANDOMIZE		乱数列の初期化
	RND	(関数)	乱数の発生

5-10

データ型の変換

数値データ間の型変換，数値データ↔文字データ間の型変換，マイクロソフト・バイナリ形式↔IEEE形式の変換を行うための関数です。

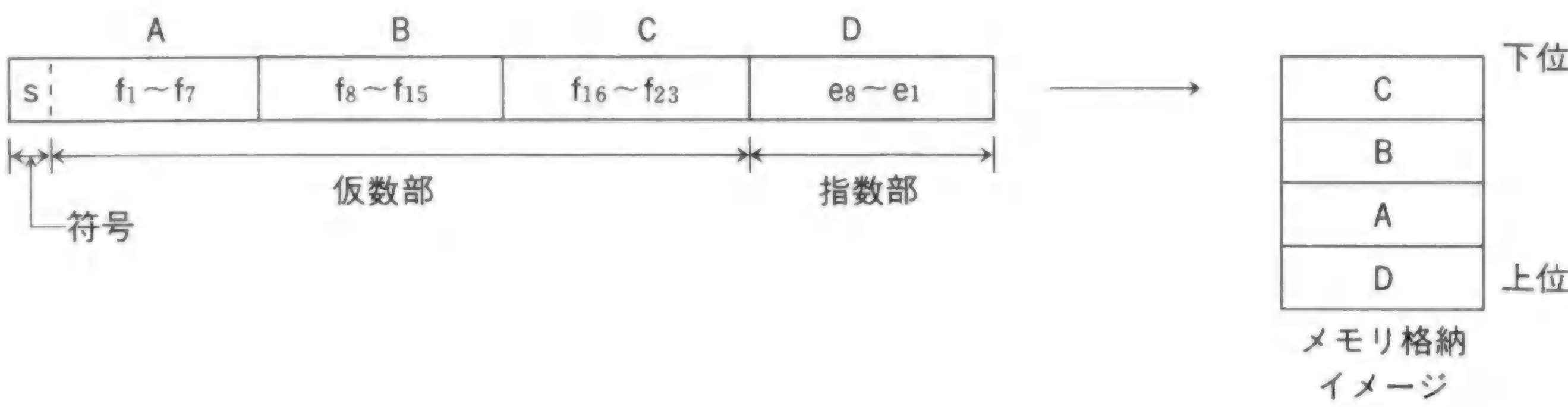
	関数	機能
数値	CINT CLNG CSNG CDBL	式の値を整数型に変換 式の値を倍長整数型に変換 式の値を単精度実数型に変換 式の値を倍精度実数型に変換
数値↕文字変換	CVD CVI CVL CVS HEX\$ MKD\$ MKI\$ MKL\$ MKS\$ OCT\$ STR\$ VAL	8バイト文字列を倍精度実数値に変換 2バイト文字列を整数値に変換 4バイト文字列を倍長整数値に変換 4バイト文字列を単精度実数値に変換 数値を16進文字列に変換 倍精度実数を8バイト文字列に変換 整数を2バイト文字列に変換 倍長整数を4バイト文字列に変換 単精度実数を4バイト文字列に変換 数値を8進文字列に変換 数値を10進文字列に変換 文字列を数値に変換
MBF↕IEEE	CVSMBF CVDMBF MKSMBF MKDMBF	4バイトのMBF文字列をIEEE形式の数値に変換 8バイトのMBF文字列をIEEE形式の数値に変換 IEEE形式の数値を4バイトのMBF文字列に変換 IEEE形式の数値を8バイトのMBF文字列に変換

■ マイクロソフト・バイナリ形式と IEEE 形式

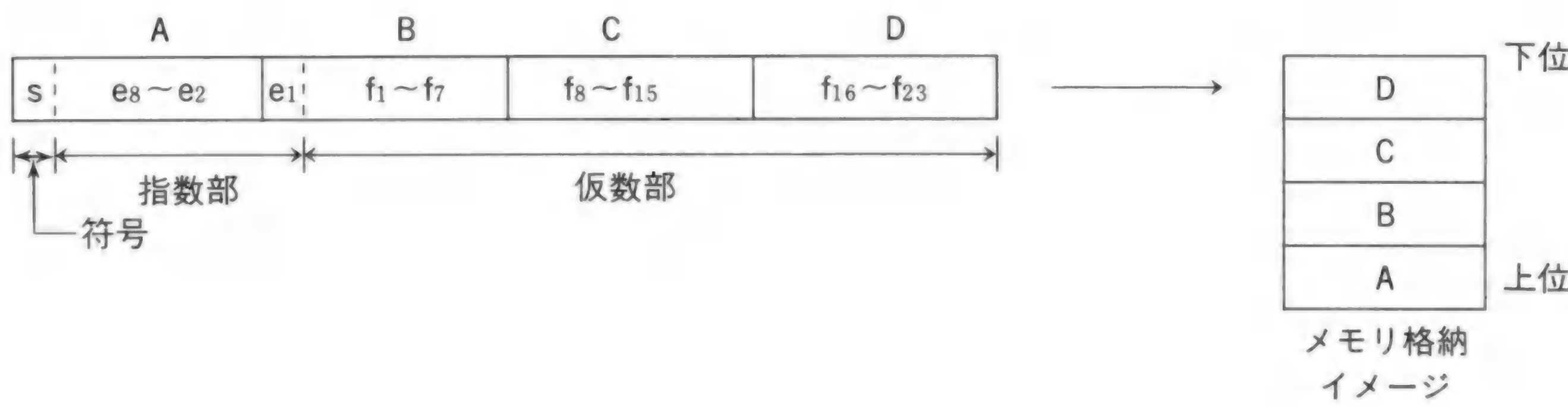
浮動小数点数の内部データ形式は、マイクロソフト・バイナリ形式と IEEE 形式とがあります。従来のマイクロソフト BASIC はマイクロソフト・バイナリ形式を用いていましたが、Quick BASIC では IEEE 形式を採用しました。IEEE 形式は、インテルの数値演算コ・プロセッサ 8087 との相性のよい標準的なデータ形式です。

以下に単精度実数のマイクロソフト・バイナリ形式と、IEEE 形式のデータ表現を示します。

● マイクロソフト・バイナリ形式



● IEEE 形式



IEEE 形式の数値を、マイクロソフト・バイナリ形式のメモリ格納イメージの文字列に変換したり、その逆を行う関数が CVSMBF, CVDMBF, MKSMBF, MKDM-BF です。

5-11

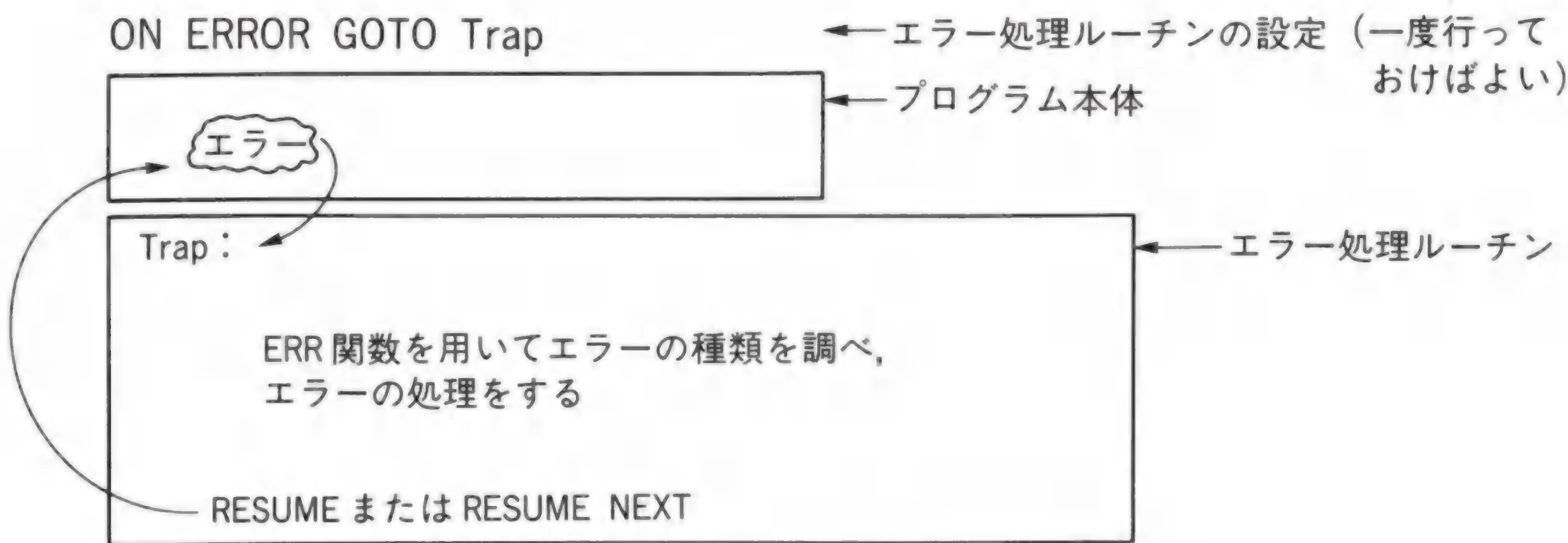
エラー処理とトラッピング

エラーが発生したときのエラー情報の取得，エラーが発生したときに分岐するトラッピングルーチンの設定，イベントが発生したときに分岐するトラッピングルーチンの設定を行うステートメントと関数です。

	ステートメント/関数	機能
エラー情報	ERL (関数)	エラーがあった行番号の取得
	ERR (//)	エラーコードの取得
	ERDEV (//)	デバイスのエラーコードの取得
	ERDEV\$ (//)	エラーが発生したデバイス名の取得
	ERROR	エラーコードに対応したエラーメッセージの表示
エラー・トラッピング	ON ERROR GOTO RESUME	エラー・トラッピングルーチンの設定 エラー・トラッピングルーチンからの復帰
イベント・トラッピング	ON event GOSUB event ON/OFF/STOP RETURN	イベント・トラッピングルーチンの設定 イベント・トラッピングの起動/無効/中断 イベント・トラッピングルーチンからの復帰

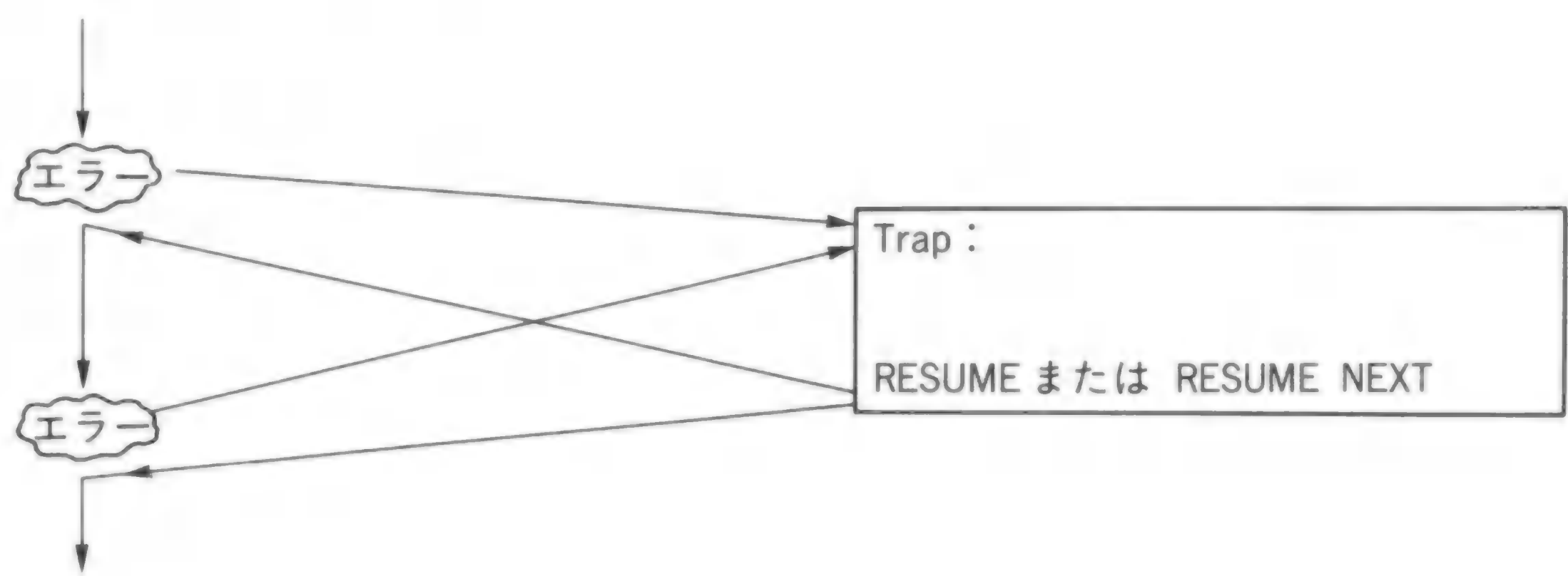
■エラー・トラッピング

プログラム実行中にエラーが発生した場合，プログラムを中断して，割り込みによりエラー処理ルーチンに分岐させることをエラー・トラッピングといいます。エラー処理ルーチンの設定は，ON ERROR GOTO 文を用いて一度行っておけば，エラーが発生するたびに，設定した処理ルーチンへ分岐します。



RESUME はエラーのあった文に戻るのに対し，RESUME NEXT はエラーのあった文の次の文に戻ります。

ON ERROR GOTO Trap (一度行っておけばよい)



■ イベント・トラッピング

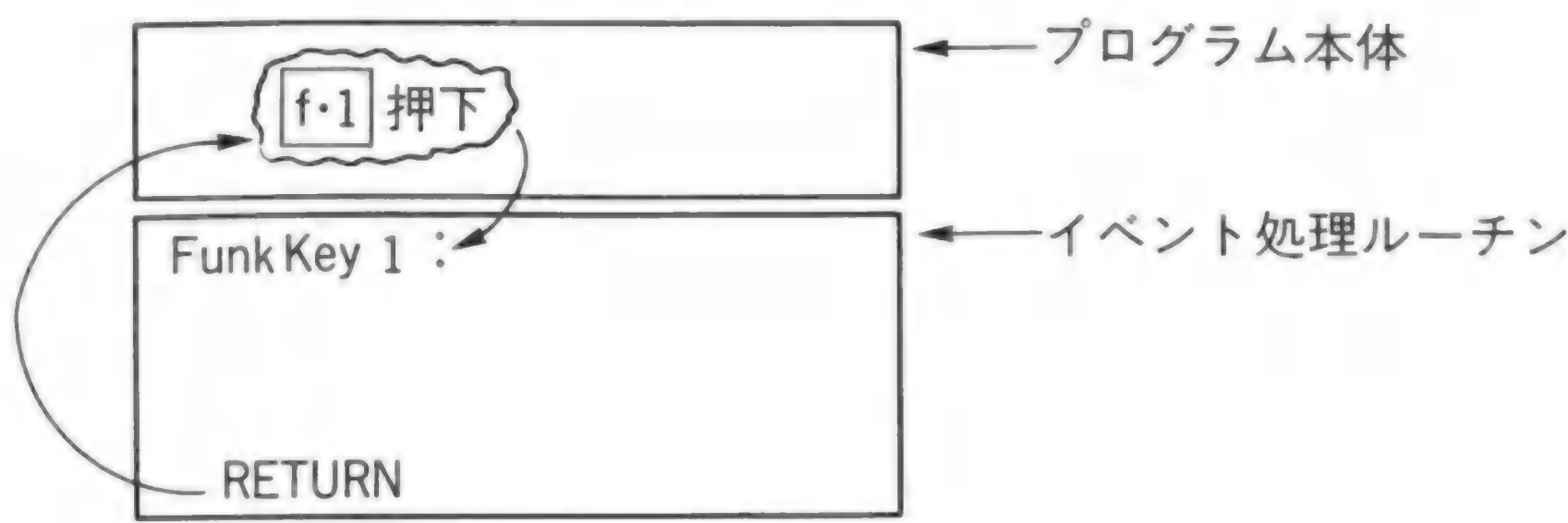
エラー・トラッピングと同様な処理を、イベントの発生により行わせることをイベント・トラッピングといいます。イベント・トラッピング処理ルーチンは、ON event GOSUB 文により設定することができます。event としては、次の3つが指定できます。

- COM(portnumber) ... シリアルポートにデータを受信したとき
- KEY(keynumber) ... 指定されたキーが押されたとき
- TIMER(interval) ... 指定秒が経過したとき

なお、イベント・トラッピングを可能にするためには、event ON によりトラッピングを有効にしておきます。

たとえば、ファンクションキーの **f.1** が押された場合の処理ルーチンは次のようになります。

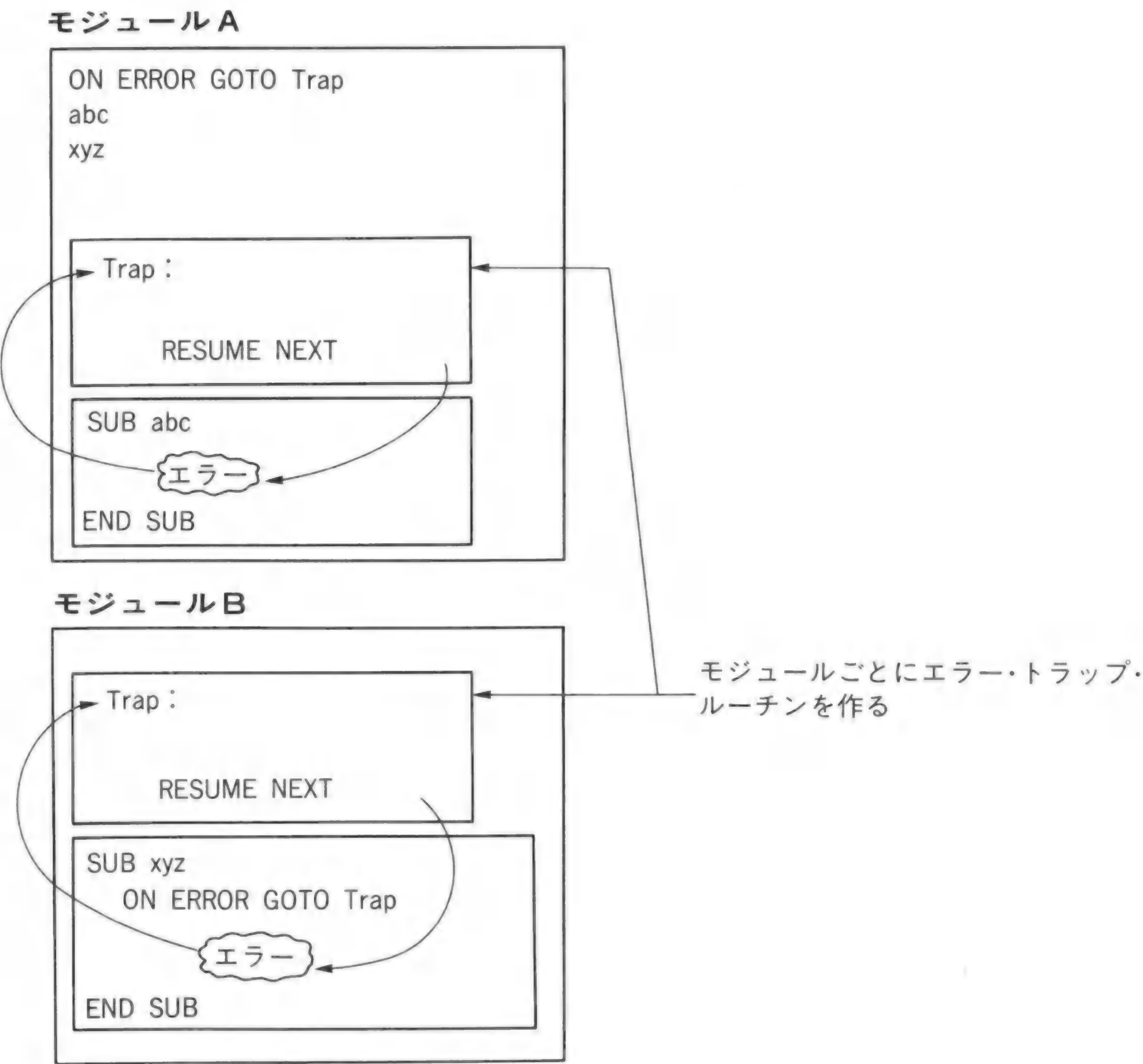
```
ON KEY (1) GOSUB FunkKey1
KEY (1) ON
```



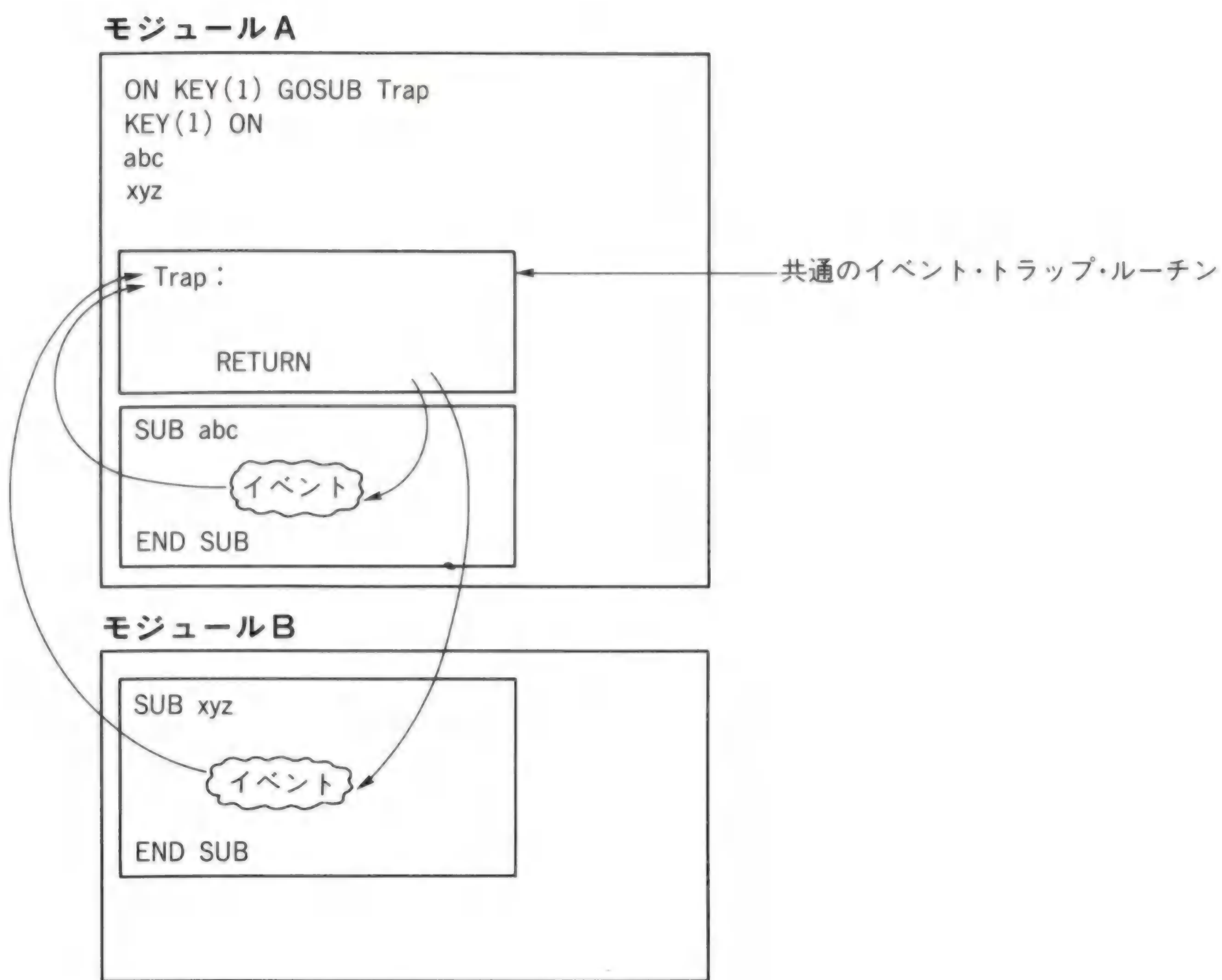
■ 複数モジュールでのトラッピング

複数モジュールでのトラッピング処理は、エラー・トラッピングとイベント・トラッピングで多少異なります。

エラー・トラッピングでは、ほかのモジュールのトラッピングルーチンを呼び出した場合、戻ってこれませんので、モジュール単位で固有のトラッピングルーチンを設定しなければなりません。



これに対し、イベント・トラッピングでは、ほかのモジュールのトラッピングルーチンを使用することができ、トラッピングルーチンはモジュール間で共通に利用されます。



■コンパイル・オプション

エラー・トラッピングを行うプログラムを、BC でコンパイルするときには次のオプションを指定します。

オプション	機能
/E	ON ERROR GOTO, RESUME line 命令がプログラム中にある場合に指定
/V /W	ON event GOSUB, event ON 命令がプログラム中にある場合に指定。/V はステートメントごとにトラッピングを行い、/W は行ごとにトラッピングを行う
/X	RESUME, RESUME NEXT, RESUME 0命令がプログラム中にある場合に指定

なお、QB 上から BC を起動する場合は、自動的にこれらのオプションが選択されます。

5-12 時間

時間の設定/取得を行ったり、タイマ・トラッピングを行うためのステートメントと関数です。

	ステートメント/関数	機 能
時間の設定／取得	DATE\$ (関数)	現在の日付の取得
	DATE\$	現在の日付の設定
	TIME\$ (関数)	現在の時刻の取得
	TIME\$	現在の時刻の設定
	TIMER (関数)	午前0時からの経過秒の取得
タイマ・トラッピング	ON TIMER GOSUB	時間経過によりサブルーチンに分岐
	TIMER ON/OFF/STOP	タイマ・イベント・トラッピングの開始/禁止/停止

5-13

メモリ操作

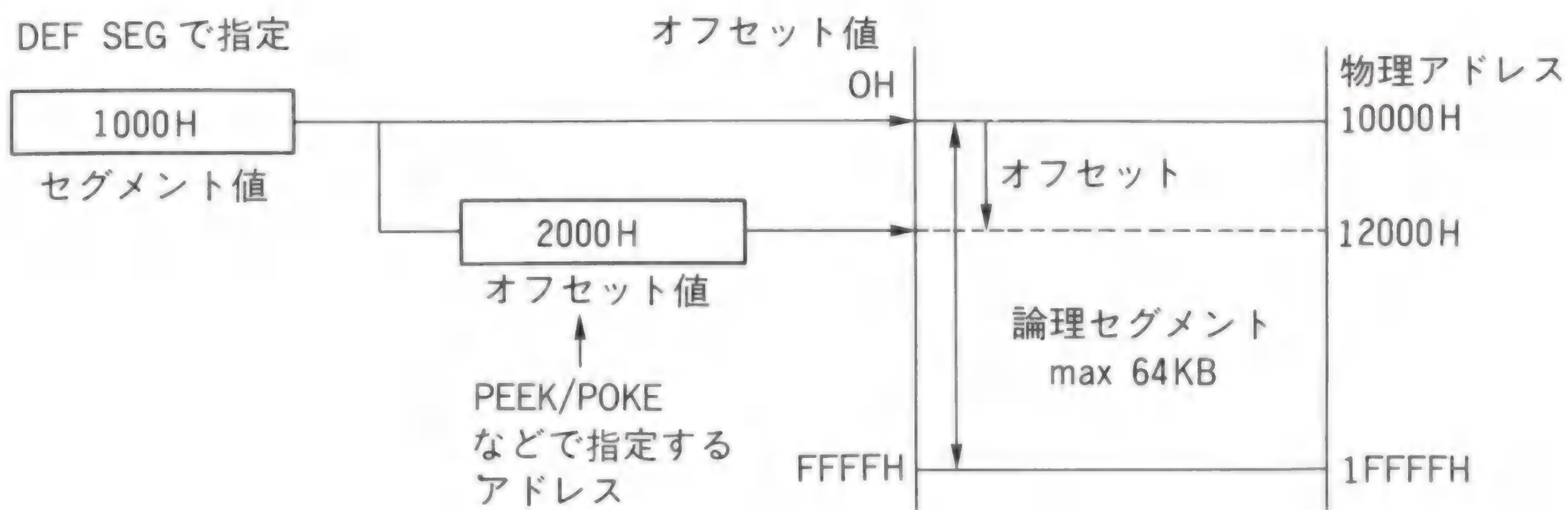
メモリのリード/ライト，メモリの取得/解放などのメモリ操作を行うためのステートメントと関数です。

ステートメント/関数	機能
DEF SEG FRE (関数) PEEK (//) POKE SETMEM	セグメント・アドレスの設定 使用可能なメモリサイズの取得 メモリから1バイトをリード メモリに1バイトをライト far ヒープで使用するメモリ量の設定

■セグメント値とオフセット値

8086では，1MB のメモリ空間を64KB のセグメントに分割して管理しています。セグメントの先頭アドレスをセグメント値といい，物理アドレスの下位4ビットを削除したものをを用います。セグメント内のアドレスは，セグメント先頭からの距離で表し，これをオフセット値といいます。

オフセット値は，0000H～FFFFH までの値をとるので，1つの論理セグメントが扱うことのできる最大メモリは，64KB となります。



メモリ・アクセスするには，まず，DEF SEG によりセグメント値を指定します。その後，PEEK, POKE, BLOAD, BSAVE, CALL ABSOLUTE で指定するアドレスは，DEF SEG で設定したセグメント先頭からのオフセット値です。

DEF SEG でセグメント値を指定しなければ，BASIC データセグメント(レジスタ DS で示される)が使用されます。

5-14

DOS 環境

MS-DOS のコマンドラインの取得，環境変数の設定/取得，システムコールなどを行うステートメントと関数です。

ステートメント/関数	機 能
CALL INT86OLD	DOS のシステムコール
CALL INT86XOLD	DOS のシステムコール
CALL INTERRUPT	DOS のシステムコール
CALL INTERRUPTX	DOS のシステムコール
COMMAND\$ (関数)	コマンドライン引数文字列の取得
ENVIRON\$ (//)	DOS 環境文字列の取得
ENVIRON	DOS 環境文字列の設定
SHELL	実行中のプログラムを一時中断して他のプログラムを実行

■ソフトウェア割り込みとシステムコール

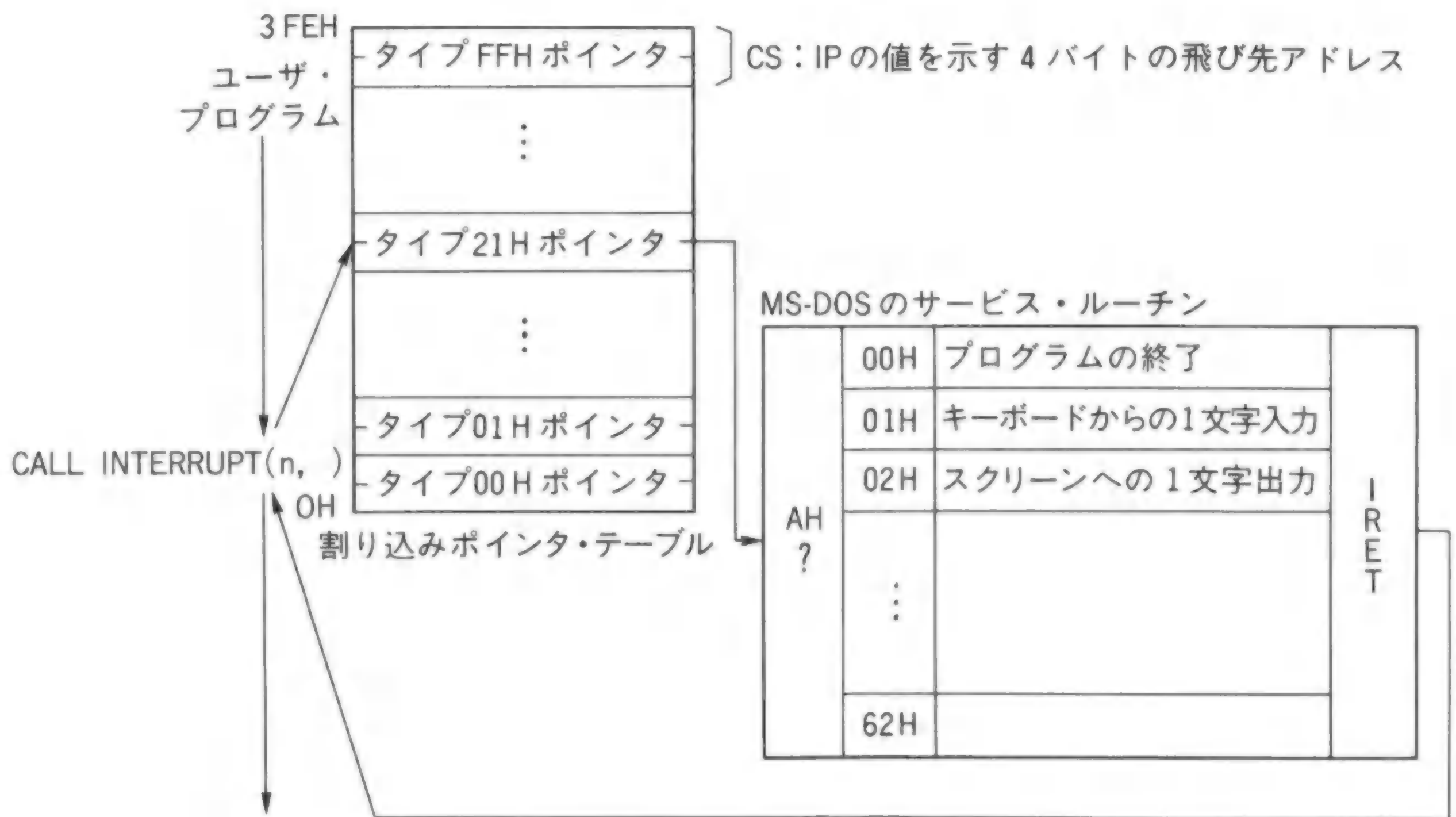
現在実行中の処理を中断して，別のサービス・ルーチンを実行したい場合に，ソフトウェア割り込みという手法を用います。

Quick BASIC からソフトウェア割り込みを行うには，

CALL INTERRUPT(n,)

を行います。n には 0 ～255を指定できます。CALL INTERRUPT(n,)が発生すると，メモリ 0 番地から設定されている割り込みポインタ・テーブルの第 n ポインタが参照され，それによって示される割り込みサービス・ルーチンへ分岐し，割り込み処理終了後，分岐してきたもとの位置に復帰します。

ソフトウェア割り込みにより，MS-DOS システムをコールするものをシステムコールといいます。システムコールの21H 番は，約90種類のファンクション(サブルーチン)を持っています。



5-15

グラフィック

グラフィック処理を行うためのステートメントと関数です。

	ステートメント/関数	機能
画面設定	CLS SCREEN WINDOW VIEW	画面のクリア 画面モードの設定 ウィンドウの設定 ビューポートの設定
画面情報	PMAP (関数) POINT (//) POINT (//)	論理座標↔物理座標の変換 描画現在位置の取得 指定位置の色の取得
表示色	COLOR PALETTE PALETTE USING	表示色の指定 パレットの設定 パレットの設定
描画	CIRCLE DRAW GET LINE PAINT PCOPY PRESET PSET PUT	円の描画 マクロコマンドによる描画 画面イメージの保存 直線の描画 閉ループ内のぬりつぶし スクリーンページを別のページにコピー 指定点のドットの消去 指定点にドットの表示 保存イメージの描画

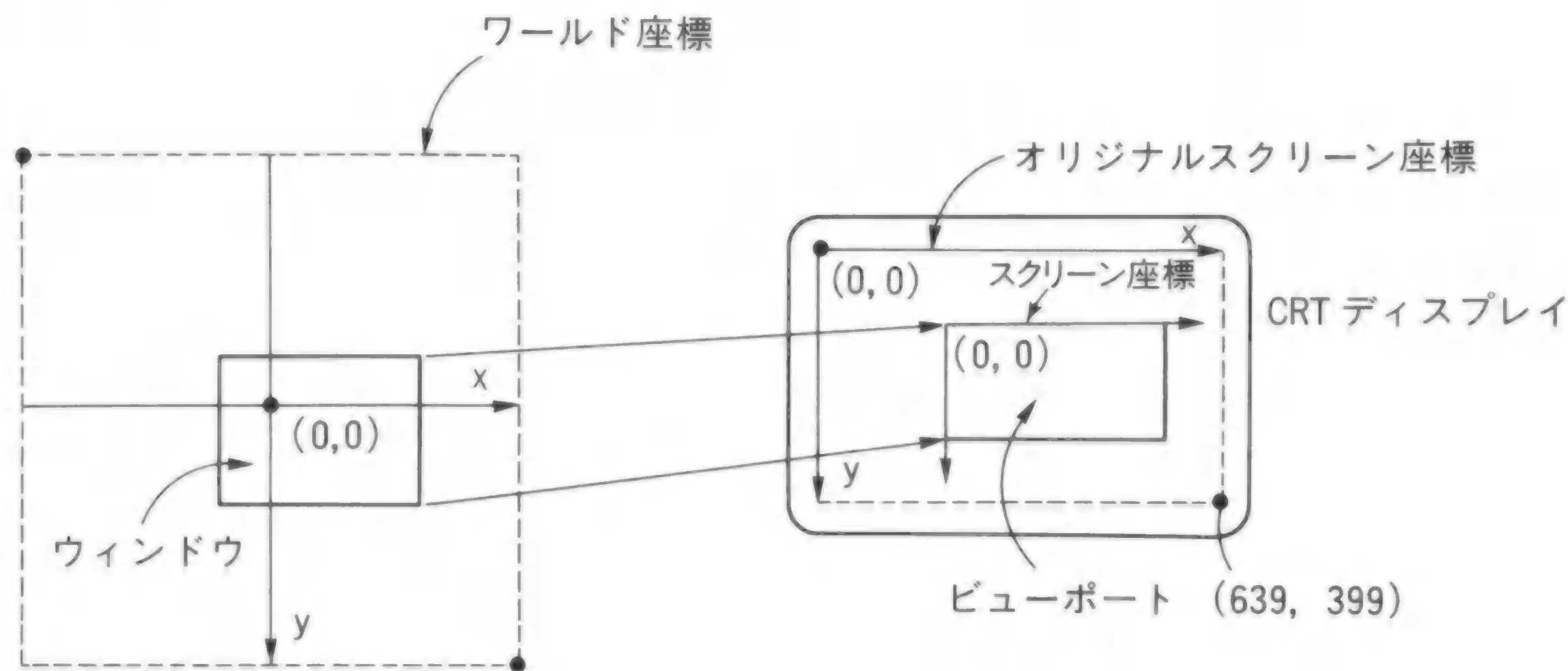
■グラフィック座標

グラフィック処理では、ワールド座標，オリジナルスクリーン座標，スクリーン座標という3つの座標を扱っています。

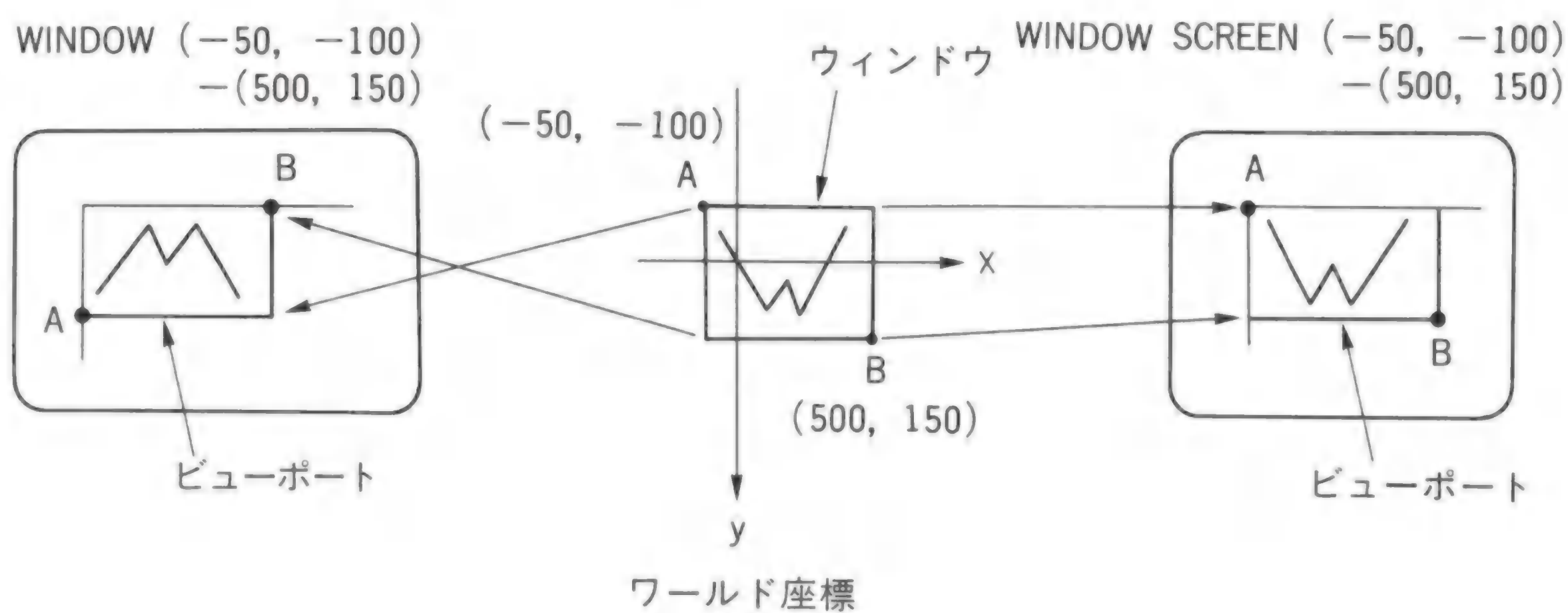
LINE や CIRCLE での作図はワールド座標に行われます。ワールド座標のどの範囲を CRT ディスプレイへの表示範囲にするかを WINDOW 文で指定します。この範囲をウィンドウと呼びます。

CRT ディスプレイ上の表示範囲は VIEW 文で指定します。この範囲をビューポートと呼びます。CRT ディスプレイの左隅上の点を (0, 0) とする座標をオリジナルスクリーン座標と呼びます。

CLS または CLS 1 で消される範囲はビューポートの中だけです。



Quick BASIC では、WINDOW 文によるウィンドウの切り方に 2 通りの方法があります。WINDOW 文に SCREEN オプションを付けると、下方を Y 座標の正の向きとするワールド座標をそのままの向きでビューポートに取り出し、SCREEN オプションを付けないと、Y 方向を逆転してビューポートに取り出します。



■画面モード

画面モードはテキスト・モード、グラフィック・モード、テキスト・モードとグラフィック・モードを同時に表示するスーパーインポーズ・モードに分かれます。さらに、グラフィック・モードはドット構成で、640×200(低解像)と640×400(高解像)に分かれます。

これらの画面モードは SCREEN 文で指定します。

SCREEN	画面モード
81	… テキスト
84	… 640×200 スーパーインポーズ
87	… 640×400 グラフィック
88または0	… 640×400 スーパーインポーズ

SCREEN 0 が実行された直後のウィンドウとビューポートは、

```
WINDOW SCREEN (0, 0)-(639, 399)
VIEW (0, 0)-(639, 399)
```

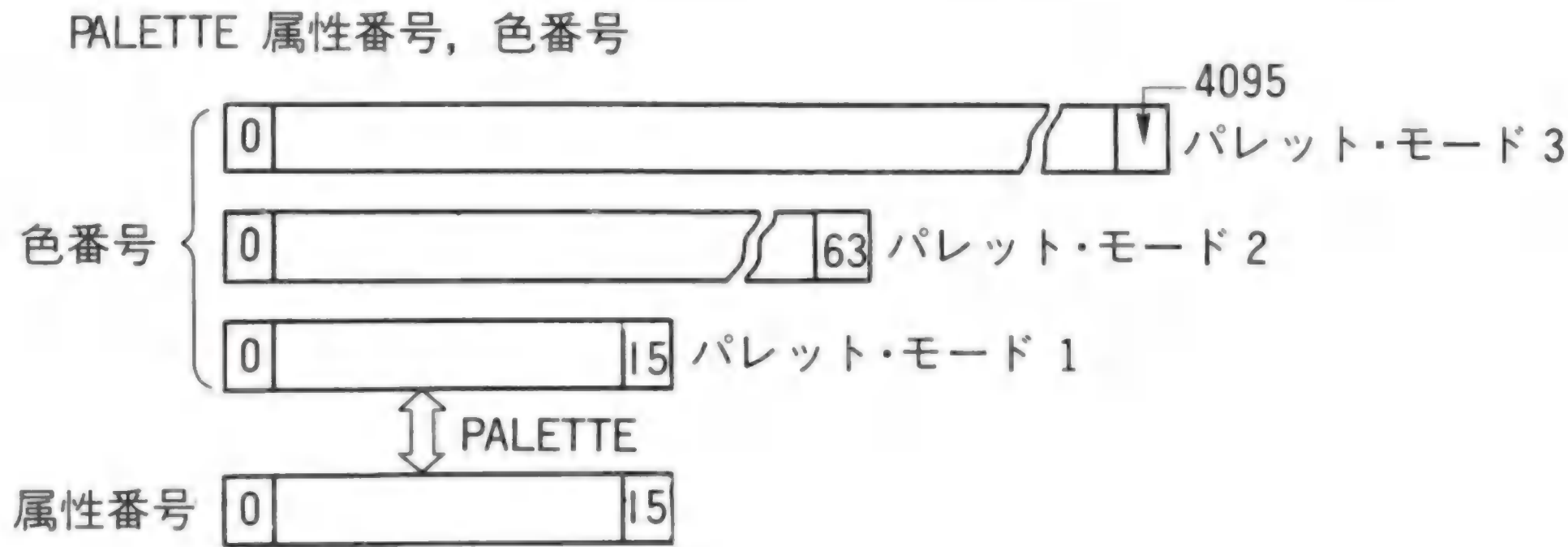
が指定されたのと同じ状態です。

■表示色

使用できる表示色の種類は SCREEN 文で指定するパレット・モードにより異なります。

SCREEN 画面モード, パレット・モード	
1 ...	16色中16色. 色番号 0 ~ 15
2 ...	64色中16色. 色番号 0 ~ 63
3 ...	4096色中16色. 色番号 0 ~ 4095

LINE や CIRCLE で指定する色は、実際の色番号ではなく、属性番号 (0 ~ 15)で行います。
属性番号と色番号は PALETTE 文により対応付けられています。



パレット・モード 1 (16色モード)のときの、色番号と色の対応は以下のとおりです。

色番号	0	1	2	3	4	5	6	7
色	黒	青	緑	水	赤	紫	黄	白

←明るい色

色番号	8	9	10	11	12	13	14	15
色	黒	青	緑	水	赤	紫	黄	白

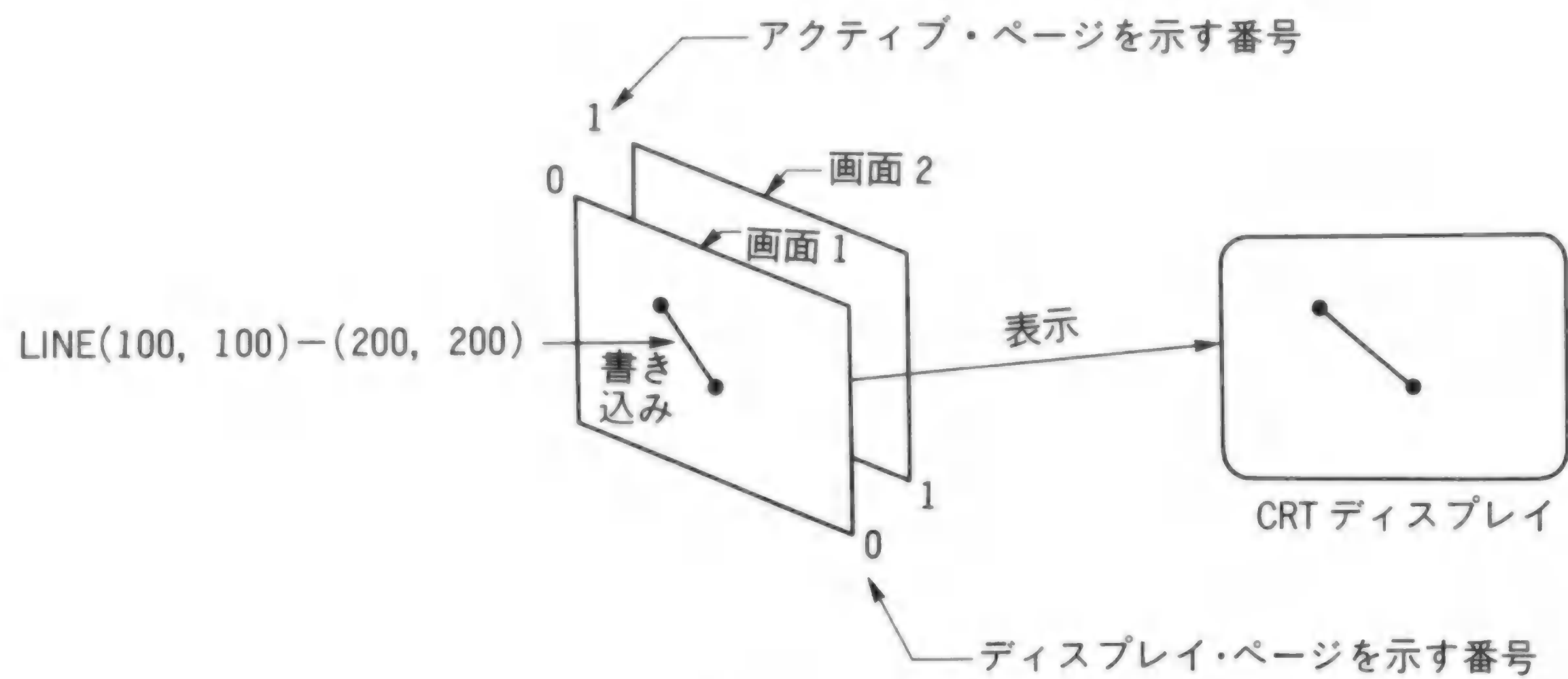
←暗い色

注) デジタル・ディスプレイでは、0 ~ 7 だけが表示できます。

■アクティブ・ページとディスプレイ・ページ

アクティブ・ページはグラフィックデータを書き込む画面、ディスプレイ・ページはCRTディスプレイに表示する画面で、SCREEN 文により指定します。

SCREEN 画面モード, パレット・モード, アクティブ・ページ, ディスプレイ・ページ



注) 640×200モードでは0～3の4画面が指定できます。

5-16

その他

いままでの分類に入らないステートメントです。

	ステートメント	機 能
一 般	BEEP LET READ～DATA REM RESTORE SWAP	ビープ音の発生 代入 プログラム中のデータを読む コメント リードデータ位置の設定 変数の内容を交換
機 械 語	BLOAD BSAVE CALL ABSOLUTE	メモリイメージのファイルをメモリにロード メモリイメージをファイルにセーブ 機械語プロシージャのコール
実 行	CHAIN RUN RESET SYSTEM TRON TROFF	プログラムのチェイン プログラムの実行 すべてのディスク・ファイルをクローズ プログラムを終了し，OSに戻る トレース・オン トレース・オフ

第6章

ステートメント・ 関数リファレンス

ABS (絶対値)		関数
書 式	ABS(x)	
機 能	引数 x の絶対値を返します。	

ASC (文字コードの取得)		関数
書 式	ASC(str) ↑ 文字列	
機 能	文字列 str の先頭 1 文字のコードを返します。その文字が ASCII 文字なら ASCII コード，漢字なら 2 バイトの漢字コードです。	
例	<pre>PRINT "A -->"; ASC("A") PRINT "亜 -->"; ASC("亜")</pre> <div> A --> 65 亜 --> 1666 </div>	

ATN (アークタンジェント)		関数
書 式	ATN(x)	
機 能	x のアークタンジェントを返します。結果は $-\pi/2 \sim \pi/2$ の範囲です。	

BEEP (ビープ音)		
書 式	BEEP	
機 能	ビープ音を発生させます。	

BLOAD (メモリイメージデータのメモリへのロード)

書式

BLOAD filename[,offset]

↑↑

↑ロードするメモリの先頭オフセット値

↑ファイル名

機能

BSAVE でセーブしたメモリイメージのファイルを、offset で示されるメモリにロードします。offset を省略すると、BSAVE でセーブするときに指定したアドレスが使われます。

BSAVE (メモリイメージデータのファイルへのセーブ)

書式

BSAVE filename,offset,n

↑↑↑

↑セーブするメモリのサイズ(バイト)

↑セーブするメモリの先頭オフセット値

↑ファイル名

機能

offset 以後の n バイトのメモリのデータを、filename で示されるファイルにセーブします。n は 0 ~ 65535 までです。

例

PC-9801シリーズのグラフィックデータは、A8000番地以後のVRAM という領域に格納されています。
このデータを BSAVE でファイルにセーブし、BLOAD で再びVRAM 上にロードする方法を示します。

```
' ---/* グラフィック画面のセーブ */---
DEF SEG = &HA800
BSAVE "b:gdata1.dat", &H0, &HFFFF
DEF SEG = &HB800
BSAVE "b:gdata2.dat", &H0, &H8000
DEF SEG = &HE000
BSAVE "b:gdata3.dat", &H0, &H8000

' ---/* グラフィック・データのロード */---
DEF SEG = &HA800
BLOAD "b:gdata1.dat", &H0
DEF SEG = &HB800
BLOAD "b:gdata2.dat", &H0
DEF SEG = &HE000
BLOAD "b:gdata3.dat", &H0
```



CALL

(プロシージャのコール)

書 式

CALL name[(arg,...)]

↑ ↑
 引数
 ↑
 プロシージャ名

機 能

name で示されるプロシージャをコールします。DECLARE 文によりプロシージャ name が宣言されていれば、CALL を付けずにプロシージャ名を書くだけで、プロシージャをコールすることができます。

CALL/CALLS

(他言語プロシージャのコール)

書 式

CALL name [[{BYVAL | SEG}] arg,...]

↑ ↑ ↑
 プロシージャ名 引数渡しの方法 引数

CALLS name[(arg,...)]

↑ ↑
 引数
 ↑
 プロシージャ名

機 能

他言語プロシージャをコールします。引数の前に BYBAL または SEG を指定して、引数渡しの方法を指定することができます。指定しなければ、near 参照により引数が渡されます。

BYVAL ... 値による渡し

SEG ... far 参照

CALLS では、引数は far 参照で渡されます。

CALL ABSOLUTE (機械語プロシージャのコール)

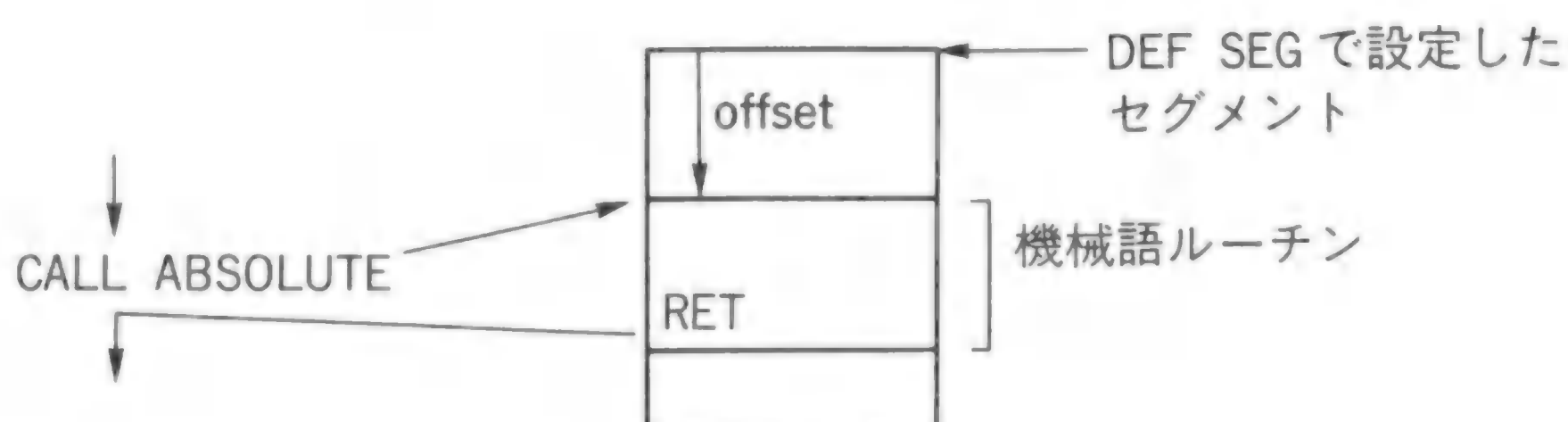
書 式

CALL ABSOLUTE([arg,...,] offset)

↑ ↑ ↑
 引数 プロシージャが置かれている
 アドレス(オフセット値)

機 能

offset で示されるメモリに置かれている、機械語ルーチンをコールします。セグメント値は、DEF SEG を用いて設定しておいてください。



CALL INT86OLD/CALL INT86XOLD
(DOS システムコール)

書 式 CALL INT86OLD/INT86XOLD(n, inarray(), outarray())

↑ システムコール後に返されるレジスタの値を入れる配列
↑ レジスタに渡すデータを入れた配列
↑ 割り込み番号

機 能 割り込み番号 n のソフトウェア割り込みを行います。inarray()と outarray()は、レジスタとの間のデータ授受を行うための整数配列で、次のようにレジスタと対応しています。

		レジスタ
inarray(0)		AX
(1)		BX
(2)		CX
(3)		DX
(4)		BP
(5)		SI
(6)		DI
(7)		FLAGS
(8)		DS
(9)		ES

INT86XOLD のときだけ使用する

INT86OLD と INT86XOLD の違いは、レジスタ DS, ES のデータ授受が行えるか否かということです。

INT86OLD と INT86XOLD プロシージャは、クイックライブラリ (QB.QLB)、およびライブラリ (QB.LIB) に含まれていますので、QB 環境で使用するには QB 起動時に、

QB /L

として、クイックライブラリ (QB.QLB) を組み込んでおかなければなりません。

例

```

' ---/* システム時間の取得 */---

DIM inarray%(9), outarray%(9)

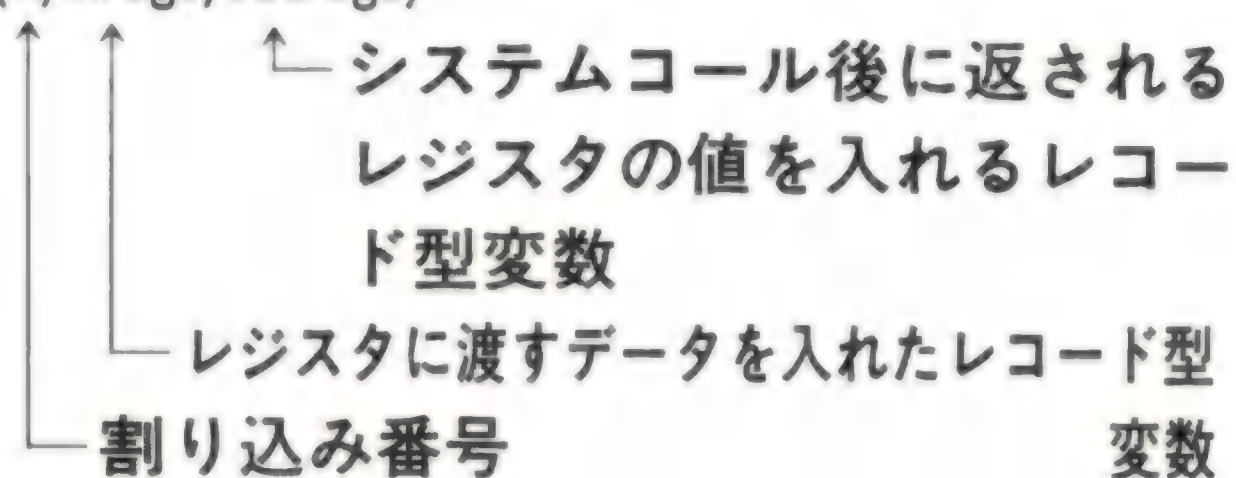
inarray%(0) = &H2C00:  時間の取得
CALL INT86OLD(&H21, inarray%(), outarray%()) ] ← システムコール

IF (inarray%(7) AND &H1) = 0 THEN ← システムコールにエラーが発生すると
    h% = outarray%(2) \ &H100:  ' 時 ← FLAGSの最下位ビットがセットされる
    m% = outarray%(2) AND &HFF:  ' 分 ← CX の上位バイトを取り出す
    s% = outarray%(3) \ &H100:  ' 秒 ← CX の下位バイトを数り出す
    PRINT "Time="; h%; ":"; m%; ":"; s%
ELSE
    PRINT "Time error"
END IF
```


CALL INTERRUPT/CALL INTERRUPTX (DOSシステムコール)

書 式

CALL INTERRUPT/INTERRUPTX(n, inregs, outregs)



機 能

割り込み番号 n の、ソフトウェア割り込みを行います。inregs と outregs は、レジスタとの間のデータ授受を行うためのレコード型変数で、その型 RegType は QB.BI の中で型宣言されています。

TYPE	RegType	
AX	AS	INTEGER
BX	AS	INTEGER
CX	AS	INTEGER
DX	AS	INTEGER
BP	AS	INTEGER
SI	AS	INTEGER
DI	AS	INTEGER
FLAGS	AS	INTEGER
DS	AS	INTEGER
ES	AS	INTEGER
END TYPE		
		INTERRUPTX のときだけ 使用する

INTERRUPT と INTERRUPTX の違いは、レジスタ DS, ES のデータ授受が行えるか否かということです。

INTERRUPT と INTERRUPTX プロシージャは、クイックライブラリ (QB.QLB)、およびライブラリ (QB.LIB) に含まれていますので、QB 環境で使用するには、QB 起動時に、

QB /L

として、クイックライブラリ (QB.QLB) を組み込んでおかなければなりません。

CALL INT86OLD/CALL INT86XOLD と CALL INTERRUPT/CALL INTERRUPTX は機能的には同じものですが、レジスタ引数に前者は配列、後者はレコード型を用いている点が異なります。データをスマートに記述するためには、後者を用いるのがよいでしょう。

例 システムコール(21H)のファンクション2CH を用いて、システム時間を取得するものです。

```
' $INCLUDE: 'qb.bi'  ← RegType 型がここで宣言されているので必ず取り込む

DIM InRegs AS RegType, OutRegs AS RegType

InRegs.ax = &H2C00: ' 時間の取得
CALL INTERRUPT(&H21, InRegs, OutRegs)  ← システムコール

IF (OutRegs.flags AND &H1) = 0 THEN  ← システムコールにエラーが発生すると
    h% = OutRegs.cx Y &H100: ' 時  ← flagsの最下位ビットがセットされる
    m% = OutRegs.cx AND &HFF: ' 分  ← CX の上位バイトを取り出す
    s% = OutRegs.dx Y &H100: ' 秒  ← CX の下位バイトを取り出す
    PRINT "Time="; h%; ":"; m%; ":"; s%
ELSE
    PRINT "Time error"
END IF
```

Time= 10 : 39 : 54

CDBL (式の値を倍精度定数型に変換) 関数

書 式

```
CDBL(x)
      ↑ 式
```

機 能

x で示される式の値を、倍精度実数値に変換します。

CDBL\$ (1 バイト文字を2 バイト文字に変換) 関数

書 式

```
CDBL$(str)
      ↑ 文字列
```

機 能

str で示される文字列中の1 バイト文字を、2 バイト文字に変換します。

例

```
a$ = "Quick 123 !"
PRINT a$; "---->"; CDBL$(a$)
```

Quick 123 !----> Q u i c k 1 2 3 !

CHAIN

(プログラムのチェーン)

書 式

CHAIN "filename"

↑ チェインするファイル

機 能

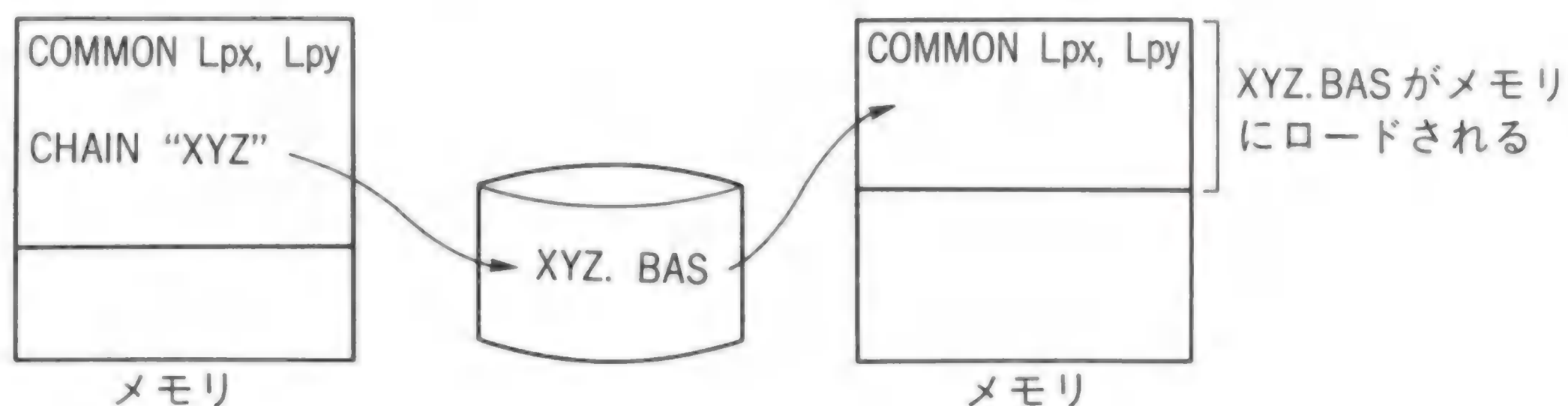
現在実行中のプログラムをメモリ上から解放し、filenameで示されるファイルをロードし、制御を移します。

QB上でfilenameの拡張子を省略すると、.BASとみなされ、コマンドラインでの実行では.EXEとみなされます。

変数の値を、チェインするプログラムに引き渡すには、COMMON宣言しておきます。

.EXEファイルでチェインする場合、BCOM42A.LIB(スタンドアロン型)はCOMMONをサポートしないので、COMMONを使用する場合は、BRUN42A.LIB(ランタイム分離型)を使用してください。

Ver.4.5ではBCOM45A.LIBとBRUN45A.LIBとなる。



CHDIR

(カレント・ディレクトリの変更)

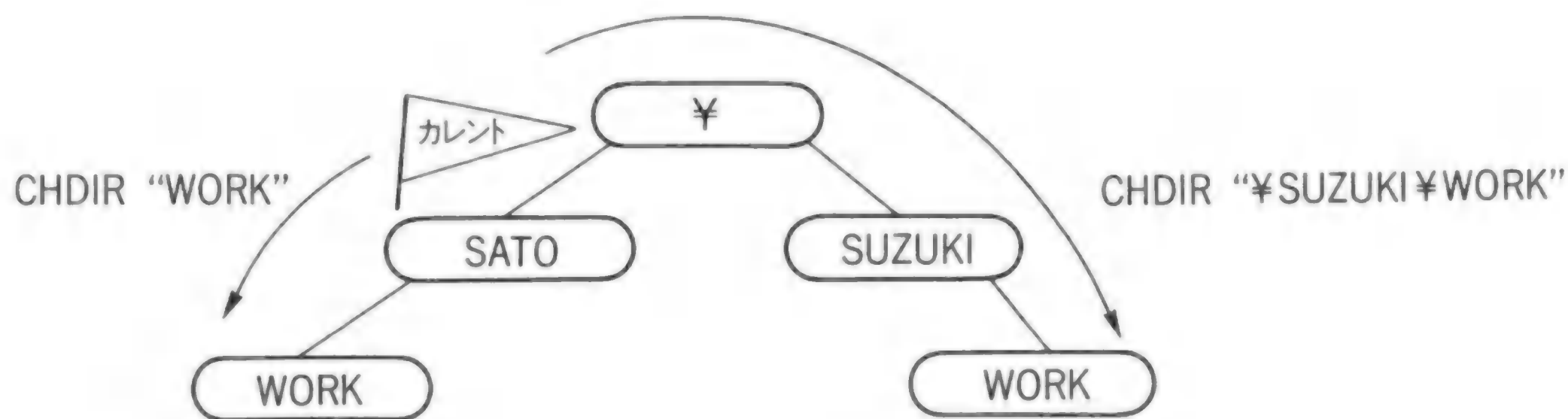
書 式

CHDIR "path"

↑パス名(サブディレクトリ名)

機 能

カレント・ディレクトリを、pathで指定したサブディレクトリに移します。



CHR\$

(ASCIIコードを文字に変換)

関数

書式

CHR\$(code[,code...])
 ↑ASCIIコード

機能

codeで示されるASCIIコード、または漢字連続コードを対応する文字に変換します。codeを複数指定した場合は、それらの文字が連結されます。CHR\$(65,1666)は“A亜”を返します。漢字連続コードについてはASCを参照ください。

例

```
CLS
FOR k = 128 TO 255
  PRINT k; CHR$(k); " ";
NEXT k
```

128	129	130	131	132	133	134	135	136	137
138	139	140	141	142	143	144	145	146	147
148	149	150	151	152	153	154	155	156	157
158	159	160	161	162	163	164	165	166	167
168	169	170	171	172	173	174	175	176	177
178	179	180	181	182	183	184	185	186	187
188	189	190	191	192	193	194	195	196	197
198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217
218	219	220	221	222	223	224	225	226	227
228	229	230	231	232	233	234	235	236	237
238	239	240	241	242	243	244	245	246	247
248	249	250	251	252	253	254	255		

CINT

(式の値を整数型に変換)

関数

書式

CINT(x)
 ↑式

機能

xで示される式の値を四捨五入して整数値に変換します。

例

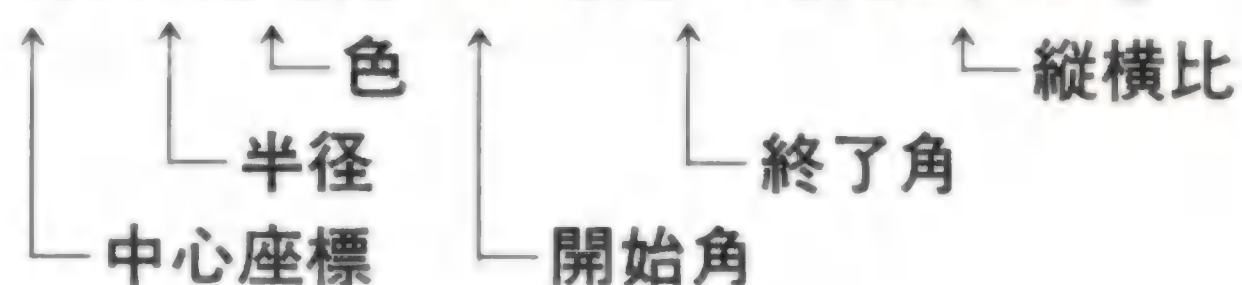
INT 参照。

CIRCLE

(円の描画)

書 式

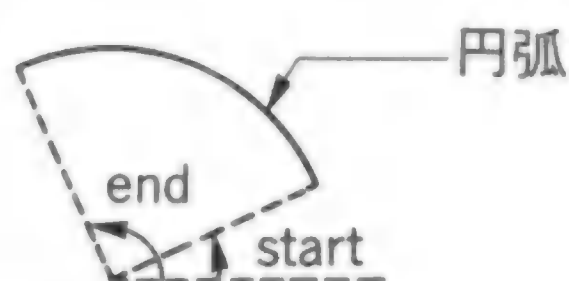
CIRCLE [STEP](x,y),r,[c],[start],[end],[aspect]



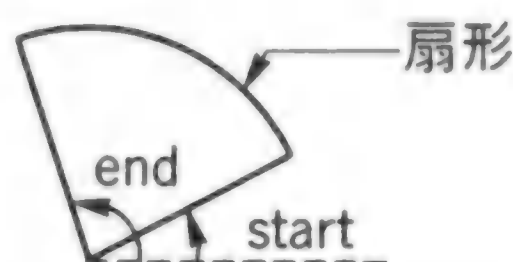
機 能

中心(x,y)で、半径rの円を色cで描きます。色cを省略すると、COLORで指定されている色となります。

startとendは、円の開始角と終了角でラジアンで指定します。



startとendに負の値を指定すると、絶対値をとった値で円弧を描き、中心と始点、中心と終点を結びます。つまり次のような扇形を描きます。



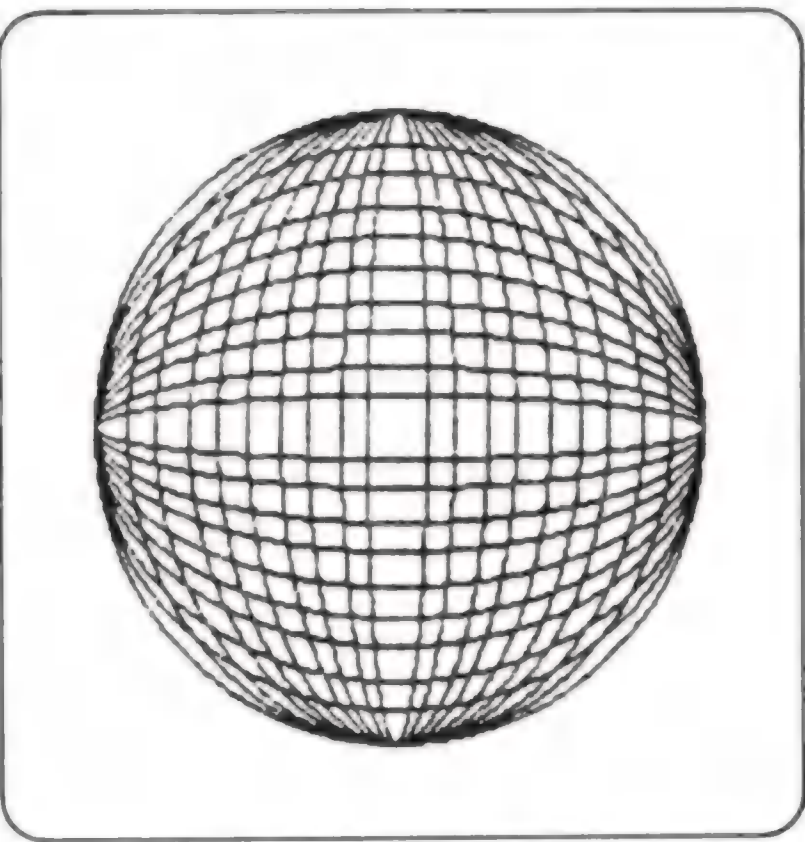
aspectは縦横比で、aspectが1より小さい場合は、x軸方向が長軸となり、1より大きい場合は、y軸方向が長軸になります。rの値が長軸となります。

(x,y)の前にSTEPを指定すると、(x,y)は絶対座標ではなく、現在位置(LP位置)からの相対座標となります。

CIRCLE文実行後の現在位置(LP位置)は円の中心に移ります。

例

```
· ---/* 楕円 */---  
  
SCREEN 0: CLS  
FOR k = 1 TO 10  
  CIRCLE (320, 200), 100, . . . k / 10  
  CIRCLE (320, 200), 100, . . . 10 / k  
NEXT k
```



CLEAR (変数の初期化)

書 式 CLEAR [, , stack]
 ↑ スタックサイズ

機 能 すべての変数をクリア(数値変数は0, 文字列変数はヌル)し, すべてのファイルをクローズします。
stack を指定すると, <デフォルトのスタックサイズ+stack>のサイズのスタックを確保します。

CLNG (式の値を倍長整数型に変換) 関数

書 式 CLNG(x)
 ↑ 式

機 能 式 x の値の小数点以下を四捨五入して, 倍長整数値に変換します。

CLOSE

(ファイルのクローズ)

書 式

CLOSE [[#]n , ...]

↑ オープンされているファイル番号

機 能

ファイル番号 n で示されるファイルをクローズします。n を省略するとオープンされているすべてのファイルをクローズします。

クローズにより、ファイル番号が解放され、出力バッファに残っているデータはファイルに書き出されます。

CLS

(画面のクリア)

書 式

CLS [{ 0 | 1 | 2 }]

↑ クリア・モード

機 能

クリア・モードにしたがって画面をクリアします。

クリア・モード	機 能
0	すべてのテキスト画面とグラフィック画面をクリア
1	グラフィック画面のビューポート内をクリア
2	画面の最下行を残して、テキスト画面のビューポート内をクリア
指定なし	グラフィック画面とテキスト画面のビューポート内をクリア。画面の最下行にファンクションキー表示を復帰させる

COLOR

(表示色の指定)

書 式

COLOR [fore][,back]

↑ ↑ バックグラウンド・カラー
↑ ↑ フォアグラウンド・カラー

機 能

フォアグラウンド・カラーとバックグラウンド・カラーを、指定した色に設定します。PC-9801シリーズでは、テキスト画面に対するバックグラウンド・カラーは指定できません。

色の値は、テキスト画面とグラフィック画面とでは意味が異なります(PC-9801シリーズの場合)。

・グラフィック画面

色の値は、属性番号(アトリビュート)を指定します。

・テキスト画面

色の値は、色番号(カラーコード)を指定します。

属性番号と色番号は、PALETTE 文により対応付けられますが、SCREEN 文実行によりデフォルト値に対応付けられます。

グラフィック画面に表示できる色の種類は、SCREEN 文で指定するパレット・モード、およびコンピュータの機種(アナログ機/デジタル機)により異なります。

SCREEN 画面モード,パレット・モード

- 1 …16色中16色.色番号0~15
- 2 …64色中16色.色番号0~63
- 3 …4096色中16色.色番号0~4095

fore,back に指定できる属性番号および色番号は、次のとおりです。

		fore	back
アナログ機	テキスト	0~31(色番号)	—
	グラフィック	0~15(属性番号)	0~15(属性番号)
デジタル機	テキスト	0~31(色番号)	—
	グラフィック	0~7(属性番号)	0~7(属性番号)

テキストに指定する色番号と色の対応は、以下のとおりです。

色番号	0	1	2	3	4	5	6	7	8~15	16~31
色	黒	青	緑	水	赤	紫	黄	白	0~7の反転	0~15の点滅

グラフィック画面に指定する属性番号と色の対応は、PALETTE 文を参照してください。

COM(通信イベントのトラッピングの許可,禁止,停止)

書 式

COM (n) {ON | OFF | STOP}

↑通信ポート番号(1または2)

機 能

nで示される通信ポートのイベント・トラッピングを次のように設定します。

- ON …許可
- OFF …禁止
- STOP …停止(停止中に受信したイベントは保管されている)

COMMAND\$ (コマンド・ライン引数文字列の取得)

関数

書 式

COMMAND\$

機 能

A>pro 1 abc xyz ↵
↓
COMMAND\$

コマンド・ライン引数のプログラム名以後の文字列を COMMAND\$ に取得します。コマンド・ラインの小文字は大文字に変換されて、COMMAND\$ に渡されます。

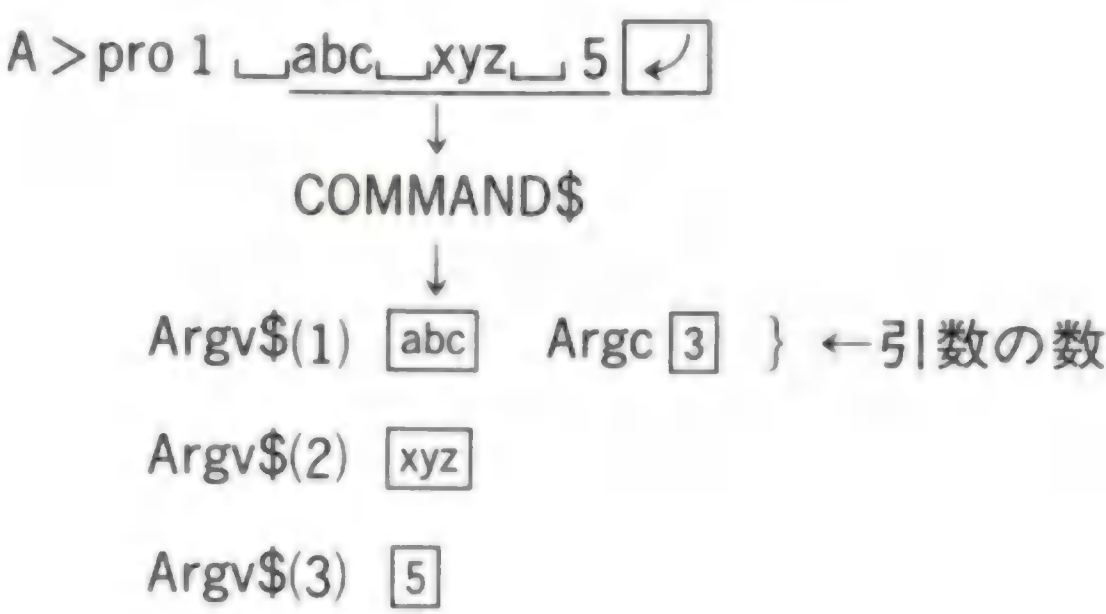
例

```
DIM Argv$(20) ← 引数を入れる文字配列

i = 1
FOR k = 1 TO LEN(COMMAND$)
  c$ = MID$(COMMAND$, k, 1)
  IF c$ <> " " THEN
    Argv$(i) = Argv$(i) + c$ ← Argv$(i)にi番目の引数が格納される
    sflag = 1 ← 空白が続く場合はiを+1しないためのフラグ
  ELSEIF sflag = 1 THEN
    i = i + 1
    sflag = 0
  END IF
NEXT k
Argc = i ← 引数の数を Argc に入れる

FOR k = 1 TO Argc
  PRINT Argv$(k)
NEXT k
```

このプログラムは、COMMAND\$ に取得したコマンド・ライン引数文字列を空白ごとにとり出し、Argv\$(1) ,Argv\$(2)…に格納し、Argc にとり出した引数の数を格納するものです。



COMMON

(共用変数の宣言)

書 式

COMMON [SHARED] [/blockname/] var [AS type], var.....

↑

COMMON ブロック名

↑

変数名

↑

型名

機 能

var で示される変数を、複数のモジュール間で共用することを宣言します。

COMMON ステートメントで宣言された変数は、特別な場所に領域を確保し、そこを複数のモジュールが共用します。このため、COMMON によって宣言される変数は、名前ではなく、宣言された順序と型に依存します。

Module 1: COMMON A, B\$

Module 2: COMMON A, C\$

Diagram shows two modules, Module 1 and Module 2, each with a COMMON block. Module 1 contains 'COMMON A, B\$' and Module 2 contains 'COMMON A, C\$'. Arrows from 'A' in both blocks point to a single shared memory location (represented by an empty box). Arrows from 'B\$' and 'C\$' point to separate memory locations (represented by two empty boxes).

モジュール 1 の B\$ とモジュール 2 の C\$ は、変数名が異なりますが、共用されています。

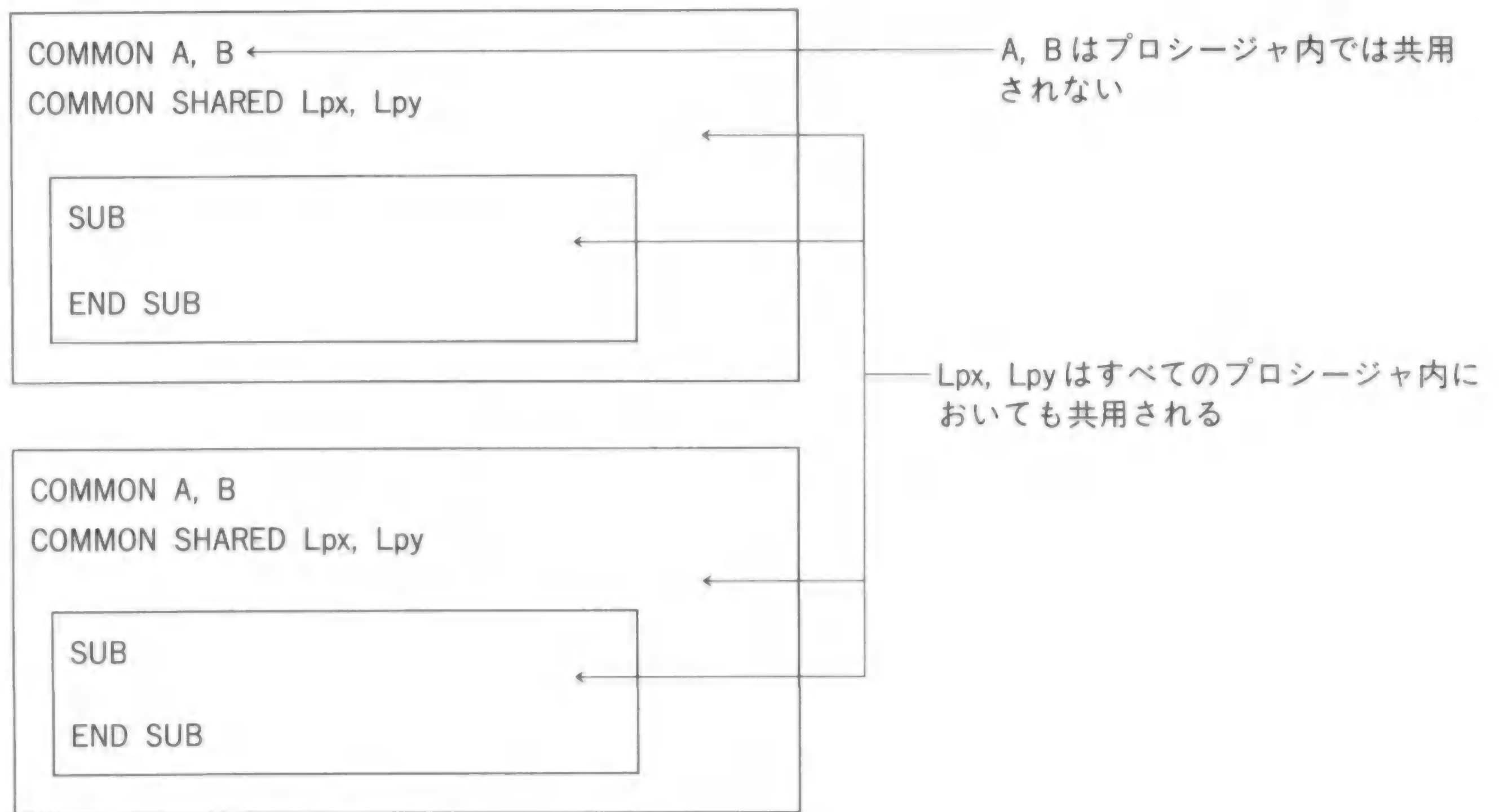
COMMON A, B\$ と COMMON C\$, A のように、2 つのモジュールで異なる型の順序で COMMON 宣言するとエラーとなります。

COMMON 文は、すべての実行ステートメントに先立って宣言する必要があります。実行ステートメントに属さないステートメントは以下のものです。

COMMON, CONST, DATA, DECLARE, DEFtyp, DIM, OPTION BASE, REM, SHARED, STATIC, TYPE~END TYPE, メタコマンド

- SHARED 属性
- SHARED 属性を付けて COMMON 宣言すると、その変数は、モジュール内のすべてのプロシージャ間で共用されます。

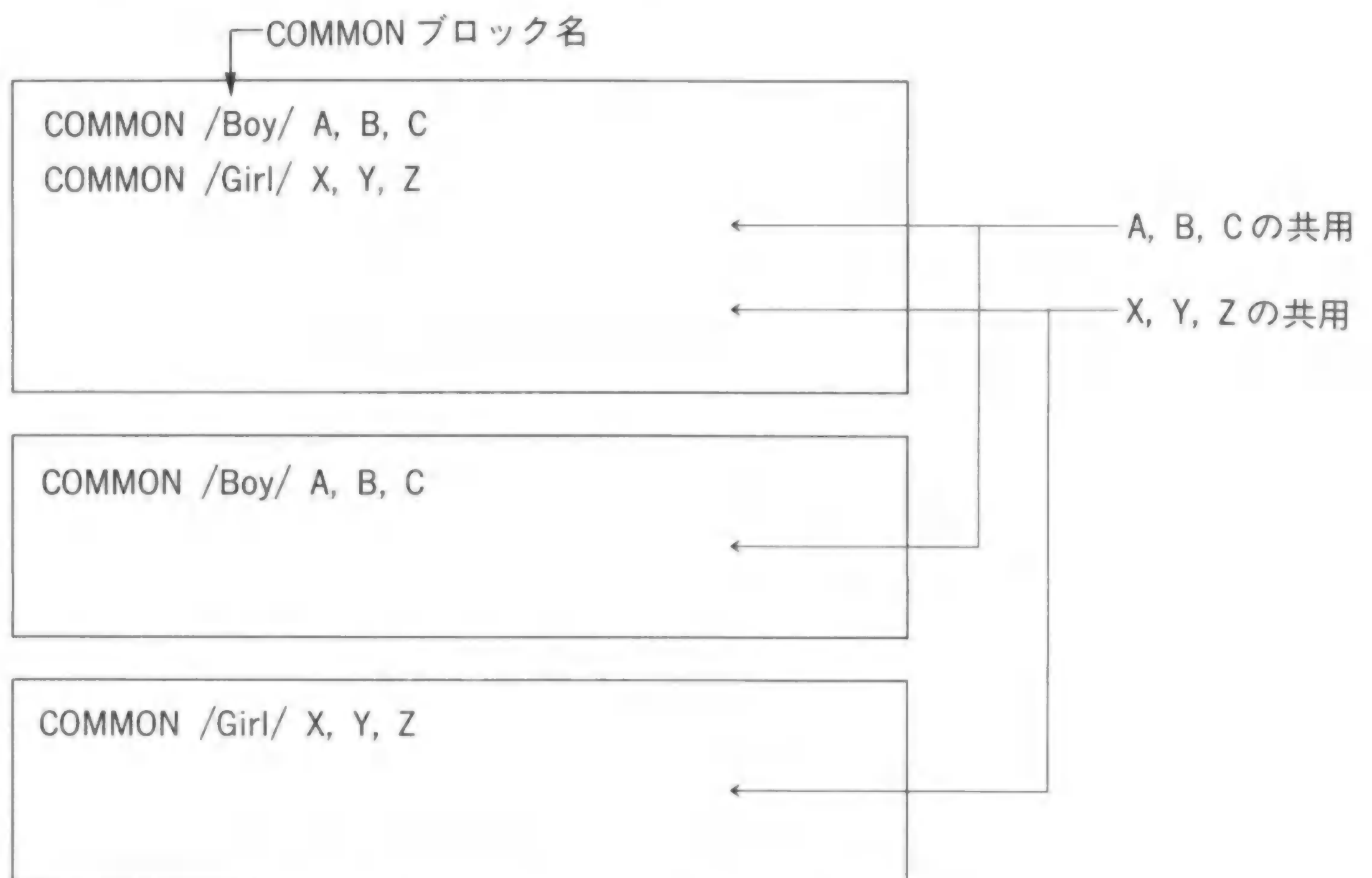
191



● COMMON ブロック名

COMMON で宣言された変数は、宣言された順に共用領域に割り当てられていきますから、すべてのモジュールで同じ順序に宣言しなければなりません。

すべての変数を共用せずにある変数だけを共用するには、COMMON ブロック名を指定します。



COMMON ブロック名を指定したものを名前付き COMMON、COMMON ブロック名を指定しないものを空白の COMMON といいます。

名前付き COMMON 変数は、CHAIN で引き継げません。

●静的配列、動的配列の COMMON 宣言

静的配列の COMMON 宣言は、次のように COMMON 宣言の前に配列のサイズを指定します。

```
DIM A(500),B(1000)
COMMON A( ),B( )
```

動的配列の COMMON 宣言は、次のように COMMON 宣言の後で配列のサイズを指定します。

```
COMMON A( ),B( )
DIM A(500),B(1000)
```

CONST

(記号定数の定義)

書式

CONST `constname = expression, ...`

↑ 式
↑ 記号定数名

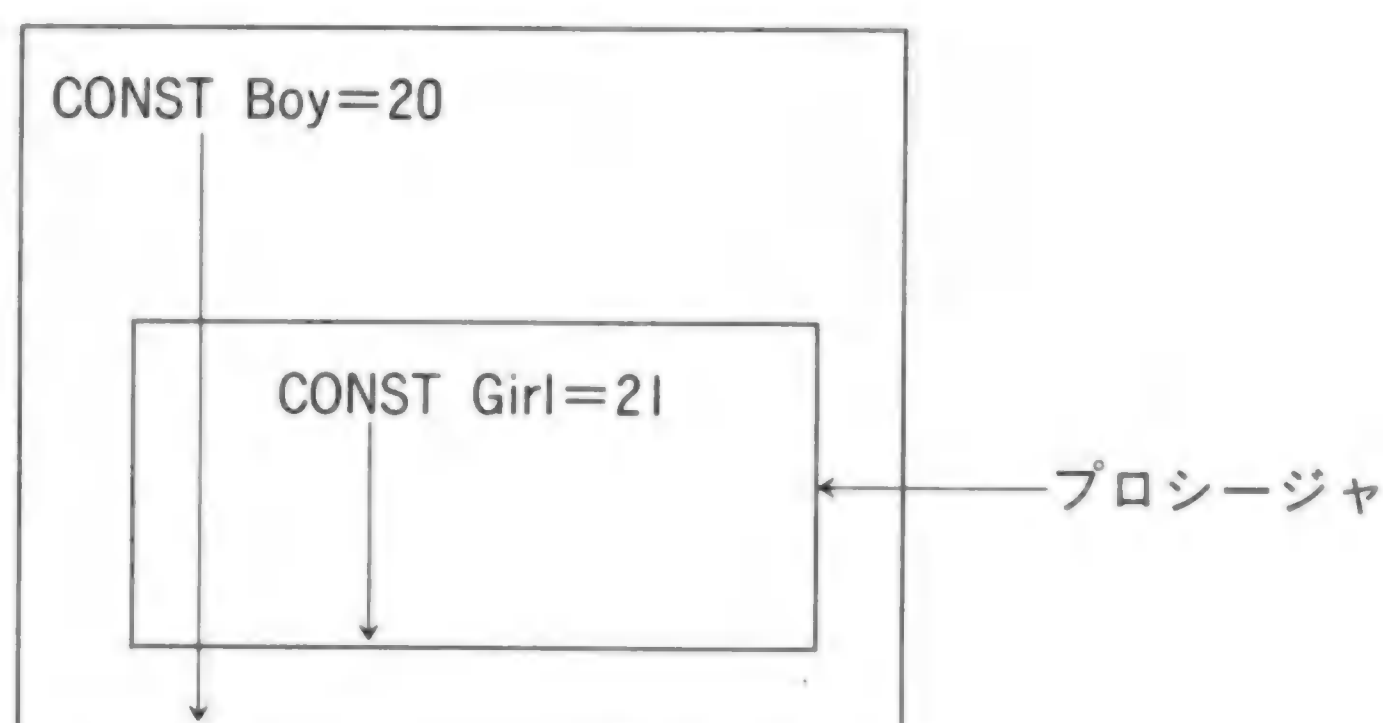
機能

constname で示される記号定数に expression で示す式の値を定義します。

式は、リテラル定数、記号定数、算術演算子(べき乗を示す[^]は含まない)、論理演算子から構成されるものです。式の中に変数や関数を含めることはできません。

記号定数を式の中に含める場合は、それが以前に定義されていなければなりません。

プロシージャ内で定義された記号定数は、そのプロシージャの中だけでローカルです。プロシージャ外で定義された記号定数は、モジュール全体でグローバルです。



記号定数名に型宣言文字（%, &, !, #, \$）を付け加えて、型を示すこともできます。このように、型宣言文字を付加して定義された記

号定数を使用する際に、型宣言文字を省略すると、定義された型とみなされます。

例	CONST	Boy = 20, Girl = 21	
	CONST	Seito = Boy + Girl	← Seito は 20 + 21 = 41 と定義される
	⋮		
	DIM	Point (Seito)	
	CONST	MaxRec % = 100	
	⋮		
	DIM	A (MaxRec)	
			↑ 整数型 (%) として扱われる
	CONST	Seito = Boy + Girl, Boy = 20, Girl = 21	
			↑ Boy と Girl が先に定義されていないのでエラー

COS		(コサイン)	関数
書 式	COS(x)		
機 能	x のコサインの値を返します。x の単位はラジアンです。		
例	SIN 参照。		

CSNG		(式の値を単精度実数型に変換)	関数
書 式	CSNG(x)		
機 能	式 x の値を単精度実数値に変換。		

CSNG\$		(2 バイト文字を 1 バイト文字に変換)	関数
書 式	CSNG\$(str) ↑ 文字列		
機 能	str で示される文字列中の 2 バイト文字を 1 バイト文字に変換します。 CDBL\$ の逆の機能です。		

CSRLIN

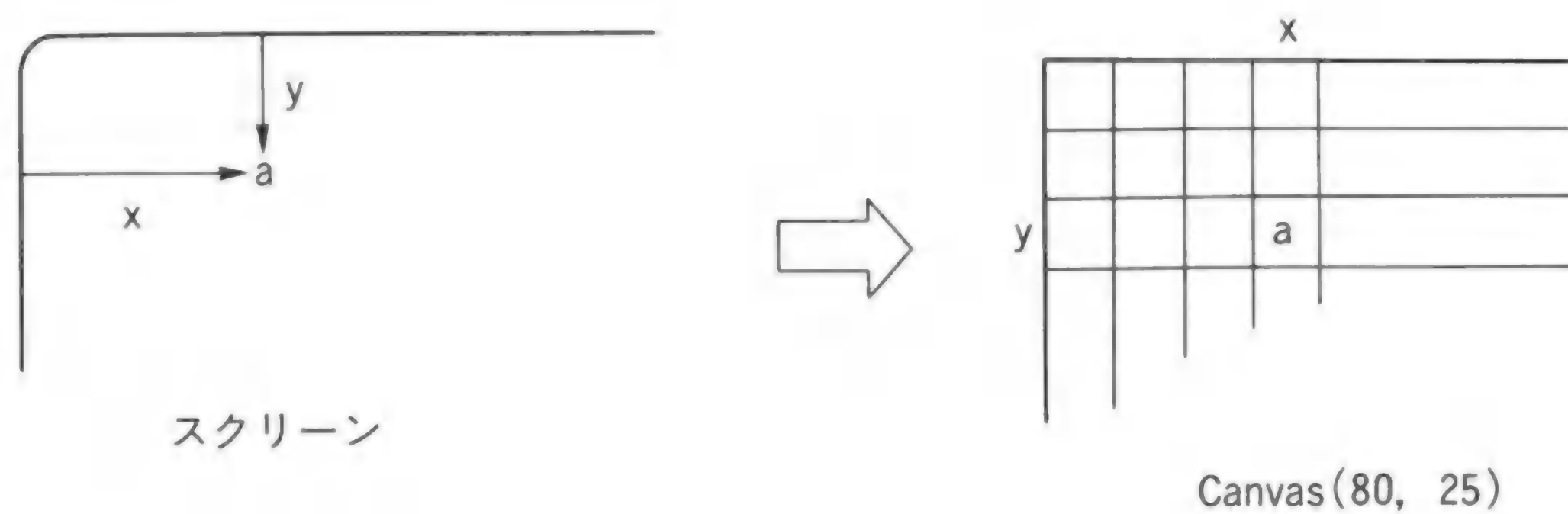
(現在のカーソル行の取得)

関数

書 式 CSRLIN

機 能 カーソルが現在位置している行を返します。

例 スクリーン上のデータを格納する配列 Canvas(80,25)を用意しておき、スクリーン上の現在位置を取得して、それに対応した配列要素に、スクリーン上の文字を格納するものです。



```
DECLARE FUNCTION GetText! ()  
DIM SHARED Canvas(80, 25) AS STRING *1
```

```
CLS  
WHILE GetText <> -1  
WEND
```

```
CLS  
FOR y = 1 TO 25  
  FOR x = 1 TO 80  
    PRINT Canvas(x, y);  
  NEXT x  
NEXT y
```

配列に取得した
データの表示

```
FUNCTION GetText  
  c$ = INKEY$  
  IF c$ <> "" THEN PRINT c$;  
  IF c$ >= " " THEN Canvas(POS(0), CSRLIN) = c$  
  IF c$ = CHR$(27) THEN GetText = -1  
END FUNCITON
```

現在位置の取得

ESC キーがデータ入力の
終わり

CVI/CSV/CVL/CVD

(内部コード形式の文字列を実際の数値に変換)

関数

書 式

CVI(2-byte-string) ← 整数値に変換
CSV(4-byte-string) ← 単精度実数値に変換
CVL(4-byte-string) ← 倍長整数値に変換
CVD(8-byte-string) ← 倍精度実数値に変換

機 能

ランダムアクセスファイルから、FIELDで定義した文字列変数にリードしたデータ(内部コード形式の文字列)を、実際の数値に変換します。

例

’ ---/* ランダム・ファイルの読み込み */---

```
OPEN "b:name.dat" FOR RANDOM AS #1 LEN = 18  
FIELD #1, 16 AS Name$, 2 AS Year$
```

```
INPUT "Rec No "; Rec
```

```
WHILE Rec > 0
```

```
  GET #1, Rec
```

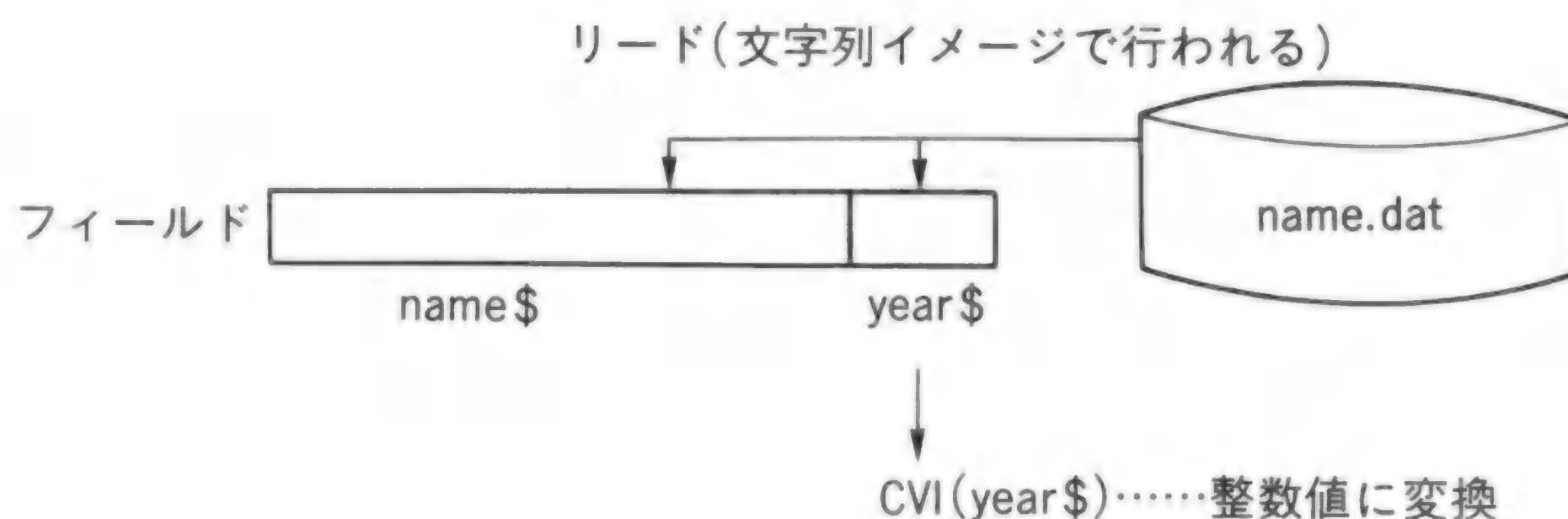
```
  PRINT Name$: CVI(Year$) ← Year$のデータを整数値に変換
```

```
  INPUT "Rec No "; Rec
```

```
WEND
```

```
CLOSE #1
```

Rec No?	1
Candy	21
Rec No?	4
Rolla	20
Rec No?	0



CVSMBF/CVDMBF (MBF 文字列を IEEE 形式の数値に変換)

関数

書 式

CVSMBF(4-byte-string) ← IEEE 形式の単精度実数値に変換
CVDMBF(8-byte-string) ← IEEE 形式の倍精度実数値に変換
↑ MBF 文字列

機 能

マイクロソフト・バイナリ形式(MBF)の文字列を、IEEE 形式の数値に変換します。

DATA

(リードするデータの定義)

書 式

DATA constant1, ...
 ↑ 定数

機 能

READ 文で読むデータを定義します。定数にはリテラル定数を指定します。記号定数を指定した場合、それは記号定数としてではなく単なる文字列としてリードされます。

文字列は二重引用符(“)で囲む必要はありませんが、カンマやコロンを文字列中に含める場合は、二重引用符で囲んでください。

データの並びの中には、ヌルデータ項目(カンマだけで区切られていてデータがないもの)を置くことができ、その項目のデータは、数値変数に読まれば0、文字列変数に読まればヌル(”)となります。

例

DATA 10,Hello,,20, 日本語, "a,b "
 ↑ ヌルデータ項目

DATES

(現在日付の取得)

関数

書 式

DATES

機 能

現在日付を、yyyy-mm-dd の10バイト文字列として取得します。

yyyy : 年 (1980～2099)
mm : 月 (01～12)
dd : 日 (01～31)

例

```
PRINT "現在の日付-->"; DATES
INPUT "年,月,日"; year$, month$, day$
DATES = year$ + "-" + month$ + "-" + day$
```

現在の日付-->1990-03-31
年,月,日? 1990,4,1

DATE\$

(日付の設定)

書 式

DATE\$ = str
↑ 日付を示す文字列

機 能

str で示す日付に現在日付を設定します。str は、yyyy-mm-dd または yy-mm-dd の文字列です。

yyyy : 年 (1980～2099)
yy : 年 (00～99)
mm : 月 (01～12)
dd : 日 (01～31)

日付は、-(ハイフン)の代わりに/(スラッシュ)で区切ってもかまいません。

DECLARE

(プロシージャの宣言)

書 式

DECLARE {FUNCTION | SUB } name (var [AS type], ...)
↑ ↑ ↑
↑ 引数の型
↑ 引数
↑ プロシージャ名

機 能

プロシージャの使用に先立って、プロシージャ名と引数の型を宣言します。

QB 統合環境において、プログラムをセーブすると、SUB プロシージャ, FUNCTION プロシージャ定義に対応した、DECLARE ステートメントが自動的に生成されます。

例

```
DECLARE SUB Disp(D$, Num)←プロシージャの宣言
:
Disp "Hello", 5←プロシージャの呼び出し
:
```

SUB	Disp(D\$, Num)
:	
END	SUB

←プロシージャの定義

DECLARE

（他言語プロシージャの宣言）

書式

The diagram illustrates the syntax of the DECLARE statement. The main text is:
`DECLARE {FUNCTION | SUB } name [CDECL] [ALIAS "aliasname "]`
`[{BYVAL | SEG }] arg [AS type], ...`
Annotations include:
- An arrow pointing to `PROCEDURE名` above `name`.
- An arrow pointing to `オブジェクト中の別名` above `"aliasname "`.
- An arrow pointing to `引数の型` above `type`.
- An arrow pointing to `引数` below `arg`.

機能

BASIC 以外の言語で書かれた外部プロシージャのプロシージャ名と引数の型の宣言をします。

CDECL… プロシージャが C 言語のコール手順を使用することを意味します。CDECL を指定すると、引数は右のものからスタックに積まれ、プロシージャ名 name は小文字に変換されて、アンダーバーが先頭に付加されます。

ALIAS ... プロシージャがオブジェクト・ファイル(ライブラリ・ファイル)中で、別の名前で登録されていることを示します。別名を `aliasname` で指定します。

BYVAL... 引数が参照(デフォルト)によってではなく、値によって渡されることを意味します。

SEG ... 引数が far 参照で渡されることを意味します。

DEF FN

(関数の定義)

書式

① DEF FNname[(arg[AS type],...)] = expression

↑ ↑ ↑ ↑
関数名 引数 引数の型 式

② DEF FNname[(arg[AS type],...)]

↑ ↑ ↑
関数名 引数 引数の型

FNname = expression

↑
式

END DEF

機能

①の書式は1論理行に収まる範囲で関数を定義でき、②の書式は複数行に渡って関数を定義できます。

関数名の先頭には必ず FN が付きます。関数の定義は呼び出しの前に行われていなければなりません。

DEF FN関数は再帰的(自分自身の定義内から自分自身を呼び出す)に定義できません。

DEF FN 関数内の変数はグローバル変数です。ローカル変数にしたい場合は、その変数を `STATIC` 宣言してください。

例

```
---/* 関数の定義 */---  
  
DEF FNHeihouwa (a, b) = a * a + b * b ← a2+b2を求める関数  
  
DEF FNMax (a, b)  
  IF a > b THEN  
    FNMax = a  
  ELSE  
    FNMax = b  
  END IF  
END DEF ← a と b の  
          大きい方を返す関数  
  
PRINT FNHeihouwa(3, 4)  
PRINT FNMax(10, 20)
```

DEF SEG

(セグメント・アドレスの設定)

書 式

DEF SEG [= address]

↑セグメント・アドレス

機 能

セグメントの先頭アドレスを、address で示されるセグメント値に設定します。BLOAD, BSAVE, CALL ABSOLUTE, PEEK, POKE で指定するオフセットは、このセグメント・アドレスからの距離です。address を省略すると、BASIC のデータセグメントが採用されます。

例

BSAVE 参照。

DEFtyp

(変数名, 関数名のインプリシット宣言)

書 式

DEFtyp letter1[- letter2], ...

↑
型

↑文字の範囲

機 能

変数名, 関数名(DEF FN 関数, FUNCTION プロシージャ)の先頭文字が、letter1~letter2 で示される文字であれば、その変数名, 関数名の型を type としてみなすように宣言します。
type として次の 5 つがあります。

INT ...整数型

LNG ...倍長整数型

SNG ...単精度実数型

STR ...文字列型

DBL ...倍精度実数型

型宣言文字(%,&,!,#,\$)は、DEFtyp 文に対してつねに優先します。

DEFtyp 文は、レコードのメンバ名には効果を持ちません。

例

DEFINT	A-Z	…すべての変数名，関数名の型は整数型となる。
DEFDBL	A, S, X-Z	…A, S, X-Zで始まる変数名，関数名の型は倍精度実数型となる。
:		
Sum	←倍精度実数型	
Small %	←整数型	

DIM

(変数, 配列の宣言)

書式

DIM [SHARED] var [(lower TO)upper, ...] [AS type], ...

↑ 変数, 配列 ↑ 添字の下限値 ↑ 添字の上限値 ↑ 変数の型

機能

変数の型宣言，配列のサイズと型の宣言をします。

配列の添字は，TO を用いて下限値と上限値を指定することもできます。指定できる添字の値は-32768～32767です。

type で指定できる型は，INTEGER，LONG，SINGLE，DOUBLE，STRING(可変，固定長)があります。

SHARED を指定すると，その変数，配列はモジュール内のすべてのプロセスで使用できます。

例

DIM A(-5 TO 5, 10 TO 20) ...配列 A()の宣言

	10	20
- 5		
5		

DIM Sum, Num AS INTEGER
 ...変数 Sum は単精度実数型, 変数 Num は整数型. 2 つとも整数型とするには, DIM Sum AS INTEGER, Num AS INTEGER と宣言する.

```
TYPE    Eisei
      Star AS  STRING * 10
      Kodo  AS  INTEGER
      Shuki AS  SINGLE
END      TYPE
DIM A AS Eisei      ...A は Eisei 型の変数
```

(くり返し)

```

① DO
  ⋮
  LOOP {WHILE | UNTIL } booleanexpression
                                     ↗ 条件式

② DO {WHILE | UNTIL } booleanexpression
  ⋮
  LOOP
                                     ↗ 条件式

```

- ①ループの後部でくり返し条件を判定する後判定反復
- ②ループの前部でくり返し条件を判定する前判定反復

WHILE は条件式が真である間ループをくり返し, UNTIL は条件式が真になるまでループをくり返します.

WHILE NOT booleanexpression と UNTIL booleanexpression およびこの逆は同じことを表します.

---/* 繰り返し */---

```

Num = 1
READ Name$(Num)
DO WHILE Name$(Num) <> "/"
    Num = Num + 1
    READ Name$(Num)
LOOP
Num = Num - 1

```

…Name\$()にデータをリードする。
WHILE 型の前判定反復。

```

Num = 0
DO
    Num = Num + 1
    READ Name$(Num)
LOOP WHILE Name$(Num) <> "/"
Num = Num - 1

```

…上と同じことを WHILE 型の後判定反復で書いたもの。

```
Num = 0
DO
    Num = Num + 1
    READ Name$(Num)
LOOP UNTIL Name$(Num) = "/"
Num = Num - 1
```

…上と同じことを UNTIL 型の
後判定反復で書いたもの.

DRAW

(マクロコマンドによる描画)

書 式

DRAW stringexpression
↑マクロコマンドを示す文字列式

機 能

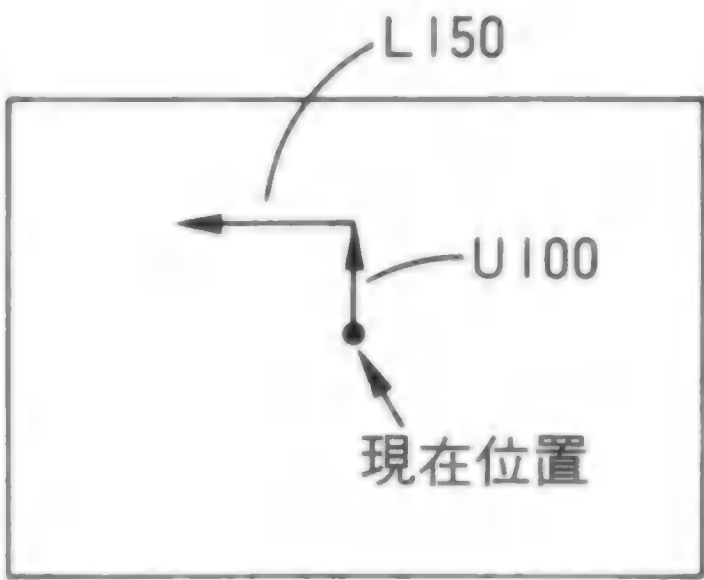
stringexpression で示されるマクロコマンドにしたがった描画をします。
マクロコマンドは以下のとおりです。

プレフィックス	B N	線を引かずに移動 移動命令の位置まで線を引き、その後(0, 0)の位置に戻る
移動	U [n] D [n] L [n] R [n] E [n] F [n] G [n] H [n] Mx,y	n単位分、上に移動 n単位分、下に移動 n単位分、左に移動 n単位分、右に移動 n単位分上, n単位分右に斜めに移動 n単位分下, n単位分右に斜めに移動 n単位分下, n単位分左に斜めに移動 n単位分上, n単位分左に斜めに移動 (x, y)位置に移動。x, yの前に+または-の符号を付けると、現在位置からの相対移動
回転・色・倍率	An TAn Cn Sn Pn,m	以後描く図形をnで示す角度(0:0°, 1:90°, 2:180°, 3:270°)だけ回転 以後描く図形をn°回転。nは-360~360 以後描く図形の色をnで示す色に設定 以後描く図形の拡大倍率をn倍に設定。nは1~255の値である。Mコマンドは相対移動のときだけ倍率がかけられ、絶対移動では倍率はかけられない 現在位置を中心に、色mで枠どられた図形の中を色nでペイント(ぬりつぶし)する
その他	"X"+VARPTR\$(str) "command="+VARPTR\$(num)	文字列変数 str に格納されているマクロコマンドを実行 command で示すマクロコマンドの引数として、数値式 num で示される値を用いる

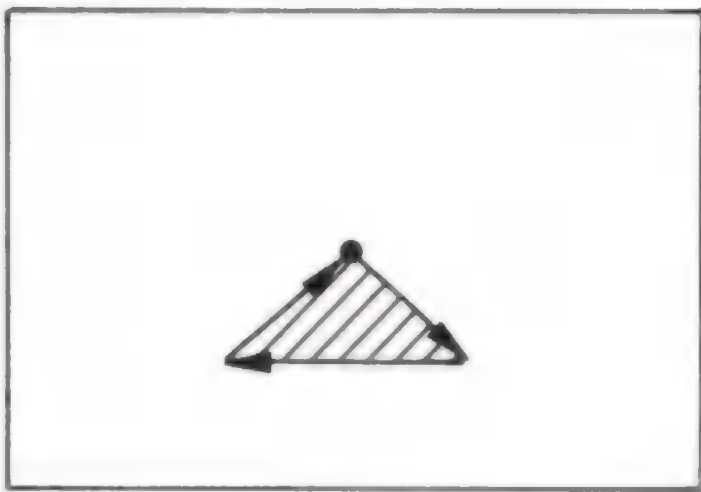
プレフィックス B,N は、次に来る移動コマンド(U~M)に対してだけ作用します。
プログラム開始時の描画現在位置は、画面の中央(320,200)です。

例

DRAW "U100 L150 "

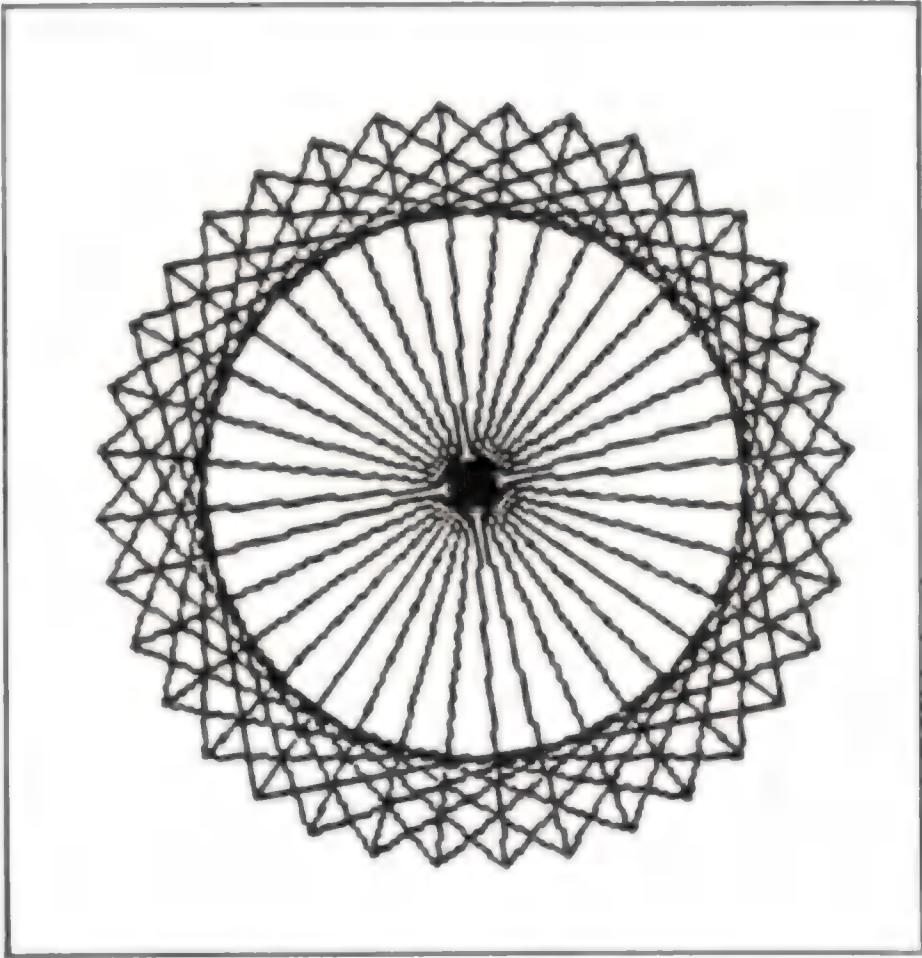


DRAW "C2" ←色をマゼンダ
DRAW "F60L120E60" ←三角形を描く
DRAW "BD60" ←三角形の内部に移動
DRAW "P1,2" ←内部をペイント



FOR A = 0 TO 360 STEP 10
DRAW "TA = " + VARPTR\$(A) + "F60L120E60"
NEXT A
...△の三角形を10°ごとに回転して描く。

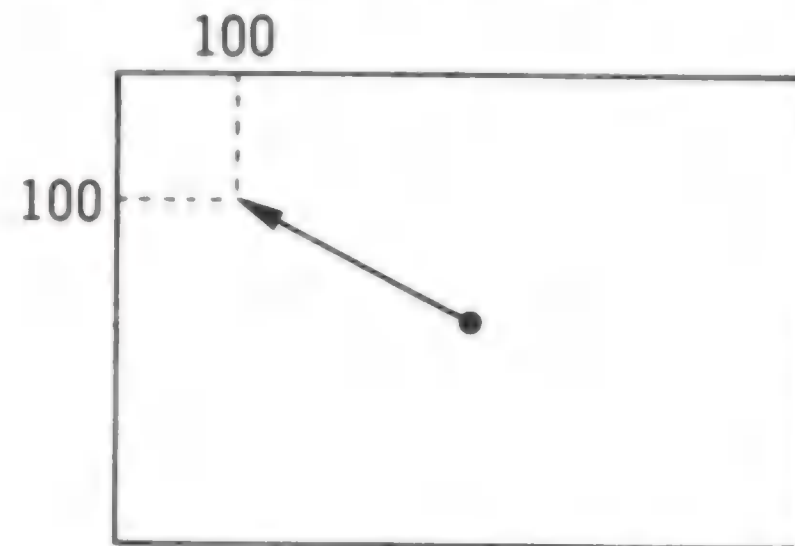
↓ 三角形の描画



A=100 : B=100

DRAW "M=" + VARPTR\$(A) + ",=" + VARPTR\$(B)

↑
各引数に対し=を付ける



END

(プログラム、プロシージャの終了)

書 式

- ① END
- ② END DEF
- ③ END FUNCTION
- ④ END IF
- ⑤ END SELECT
- ⑥ END SUB
- ⑦ END TYPE

機 能

- ①プログラムの終了。この位置でプログラムの実行を停止し、OS または QB 統合環境に制御を戻します。
- ② DEF FN 関数の定義の終わり
- ③ FUNCTION プロシージャの定義の終わり
- ④ブロック IF 文の終わり
- ⑤ SELECT CASE 文の終わり
- ⑥ SUB プロシージャの定義の終わり
- ⑦ユーザ定義型の定義の終わり

ENVIRON\$

(DOS 環境文字列の取得)

関数

書 式

↓ 環境変数名

- ① ENVIRON\$(str)
- ② ENVIRON\$(n)

↑ 環境文字列テーブル内の番号

機 能

- ① str で示される DOS 環境変数名に割り当てられている文字列を取得します。
- ②環境文字列テーブル内の n 番の環境変数名と、それに割り当てられている文字列を取得します。

環境変数が見つからなければヌル文字列を返します。文字列の大/小文字は区別されます。
つまり LIB と lib は異なる名前とみなされます。

```
ENVIRON$(LIB) ...A : ¥LIB
ENVIRON$(2) ...LIB = A : ¥LIB
```

1	PATH=A : ¥BIN ; A : ¥
2	LIB=A : ¥LIB
3	
⋮	⋮

環境文字列テーブル

例

```
' ---/* DOS 環境文字列の取得 */---
n = 1
DO WHILE ENVIRON$(n) <> ""
  PRINT n; " : "; ENVIRON$(n)
  n = n + 1
LOOP
```

…環境文字列テーブルに登録されている文字列を表示

1	: COMSPEC=A:¥COMMAND.COM
2	: PATH=A:¥BIN
3	: LIB=A:¥LIB

ENVIRON (DOS 環境文字列の設定)

書 式

```
ENVIRON "parameter = text"
           ↑           ↑
           環境変数名 文字列
```

機 能

parameter で示される環境変数に、text で示される文字列を割り当てます。この環境変数は、環境文字列テーブルの末尾に付け加えられます。もし環境変数がすでにテーブル内にあれば削除されます。
text がヌル文字列 ("") またはセミコロン (" ; ") なら、環境文字列テーブルから環境変数が削除されます。
parameter = text は parameter text と書くこともできます。

例

```
ENVIRON "PATH = A : ¥BIN ; A : ¥ "
…PATH 環境変数を A : ¥BIN ; A : ¥ に設定
```


EOF

(ファイル終わりの検知)

関数

書 式

EOF(n)

↑ ファイル番号

機 能

ファイル番号 n でオープンされているファイルの、ファイルエンドで真(-1)を返します。

EOF はデバイス・ファイル SCRN :, KYBD :, CONS :, LPTn : には使用できません。

通信デバイスをアスキーモードでオープンしている場合、**CTRL** + **Z** の受信で EOF は真(-1)となり、デバイスをクローズするまで、真の状態を保ちます。バイナリモードでオープンしている場合は、入力待ち行列が空(LOC(n)=0)であれば EOF は真となり、空でなければ EOF は偽となります。

例

INPUT\$, INPUT #, LINE INPUT # 参照。

ERASE

(配列変数の初期化と解放)

書 式

ERASE arrayname, ...

↑ 配列名

機 能

①静的配列の場合

指定した配列が静的配列の場合、配列要素は、数値配列なら 0、文字列配列ならヌル文字列に初期化されます。

②動的配列の場合

指定した配列が動的配列の場合、その配列が割り当てられているメモリが解放されます。

例

```
REM $DYNAMIC
DIM A(100,100)
:
ERASE A    ←配列 A( ) が割り当てられていたメモリを解放

REM $STATIC
DIM B(100,100)
:
ERASE B    ←配列 B( ) のすべての要素を 0 に初期化
```

ERDEV

(デバイスのエラーコードの取得)

関数

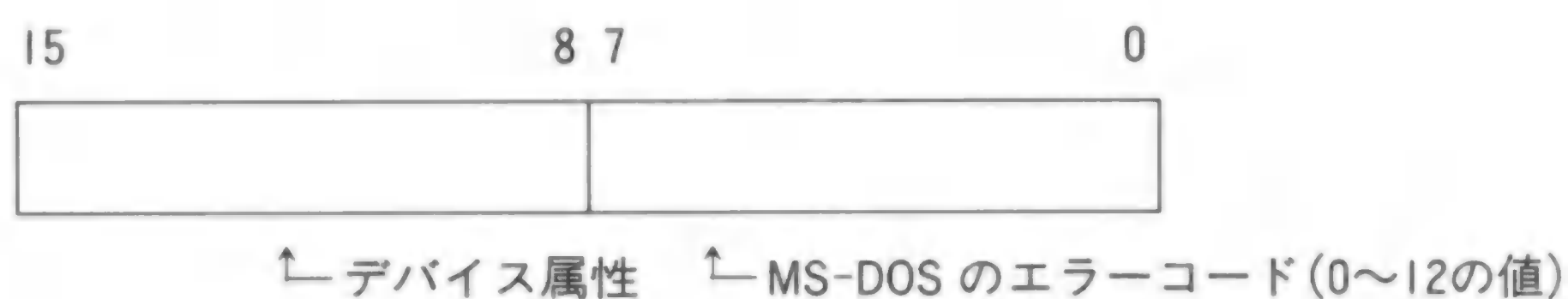
書 式

ERDEV

機 能

エラーが発生したデバイスからのエラーコードを返します。この関数は通常、ON ERROR 文で指定した、エラー処理ルーチン内で使用します。

エラーコードのビットの意味は次のとおりです。



例

```
' ---/* エラー・ハンドラ */---  
ON ERROR GOTO ErrorHandler  
  
OPEN "b:abc.qat" FOR INPUT AS #1  
  
END  
  
ErrorHandler:  
  PRINT "エラーのあったデバイス : "; ERDEV$  
  PRINT "MS-DOS エラーコード: "; ERDEV  
  RESUME NEXT
```

エラーのあったデバイス : B: MS-DOS エラーコード: 2

ERDEV\$

(エラーが発生したデバイス名の取得)

関数

書 式

ERDEV\$

機 能

エラーが発生したデバイスのデバイス名を返します。この関数は通常 ON ERROR 文で指定した、エラー処理ルーチン内で使用します。

例

ERDEV 参照。

ERL	(エラーがあった行番号の取得)	関数
書 式	ERL	
機 能	エラーが発生した行の行番号を返します。なお、その行には行番号を付けておく必要があります。この関数は通常、ON ERROR 文で指定した、エラー処理ルーチン内で使用します。	
例	ERR 参照。	

ERR	(エラーコードの取得)	関数
書 式	ERR	
機 能	発生したエラーのエラーコードを返します。この関数は通常、ON ERROR 文で指定した、エラー処理ルーチン内で使用します。	
例	<pre>' ---/* エラー・ハンドラ */--- ON ERROR GOTO ErrorHandler 100 a = LOG(-1) 110 b = 10 / 0 ↑ _____ 行番号 END ErrorHandler: ErrorNum = ERR ← エラーコード ErrorLine = ERL ← エラーのあった行番号 PRINT "Error in"; ErrorLine; " : "; SELECT CASE ErrorNum CASE 5 PRINT "引数が許される範囲ではありません" CASE 6 PRINT "計算結果がオーバーフローしました" CASE 7 PRINT "メモリが足りません" CASE 9 PRINT "配列の添字が許される範囲にありません" CASE 11 PRINT "0 で除算しました" CASE ELSE PRINT "その他のエラー" END SELECT RESUME NEXT</pre> <div>Error in 100 : 引数が許される範囲ではありません Error in 110 : 0 で除算しました</div>	

ERROR (エラーコードに対応したエラーメッセージの表示)

書 式 ERROR n
 ↑
 エラーコード

機 能 エラーコード n(0~255)に対応した、エラーメッセージを表示します。

例

ERROR 1 画面上に以下のようなエラーメッセージが表示され、ERROR 文のところで実行を止める。

対応する FOR がありません (ERR =1)

確認

EXIT (ループ、プロシージャからの脱出)

書 式 ① EXIT DEF
 ② EXIT DO
 ③ EXIT FOR
 ④ EXIT FUNCTION
 ⑤ EXIT SUB

機 能 ① DEF FN 関数から抜けます。
 ② DO~LOOP から抜けます。ループが入れ子になっている場合、そのループのすぐ外側のループに出ます。
 ③ FOR~NEXT から抜けます。ループが入れ子になっている場合、そのループのすぐ外側のループに出ます。
 ④ FUNCTION プロシージャから抜けます。
 ⑤ SUB プロシージャから抜けます。

例

’ ---/* ループからの脱出 */---

```
KeyDat = 5      ...10個のデータを先頭から調べ、KeyDat と同じデータが最初に見つかったときにループから抜ける。
FOR k = 1 TO 10
  READ a
  IF a = KeyDat THEN
    PRINT k, a
    EXIT FOR
  END IF
NEXT k

DATA 2, 3, 1, 5, 6, 0, 1, 2, 1, 1
```



```
DECLARE FUNCTION Search! (a!(), KeyDat!, n!)
' ---/* 関数からの脱出 */---

DIM a(10)
FOR k = 1 TO 10
    READ a(k)
NEXT k
DATA 2, 3, 1, 5, 6, 0, 1, 2, 1, 1

PRINT Search(a(), 15, 10)
PRINT Search(a(), 6, 10)

FUNCTION Search (a(), KeyDat, n)
    FOR k = 1 TO n
        IF a(k) = KeyDat THEN
            Search = k
            EXIT FUNCTION ← ファンクションプロシージャ Search
                             から呼び出し元に戻る
        END IF
    NEXT k
    Search = -1 ← データが見つからなかったとき
END FUNCTION
```

…Search は配列 a() の中から KeyDat を探し、
その位置を返す。

EXP

(指数)

関数

書 式

EXP(x)

機 能

指数 e^x の値を返します。x には 88.02969 以下の値を与えてください。

FIELD

(ファイルバッファの設定)

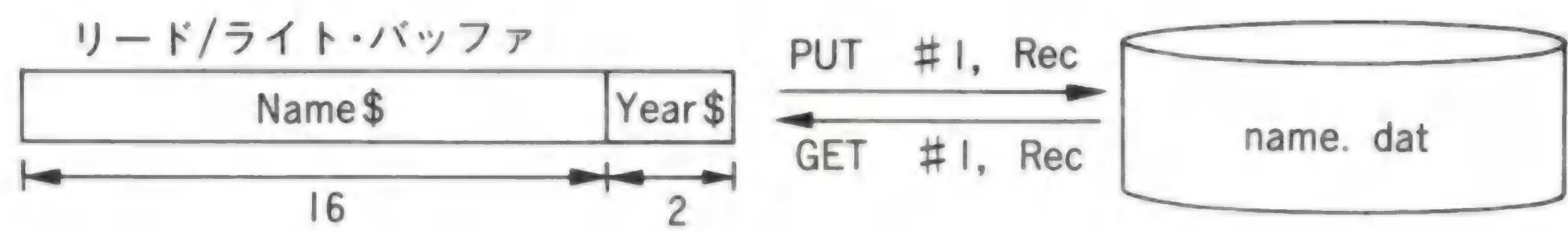
書 式

FIELD [#]n ,w AS strvar, ...
 ↑ ↑
 ↑ 文字列変数
 strvar に入れる文字列の長さ
 ↑
 ファイル番号

機 能

ランダム・ファイルとの間のリード/ライト・バッファの設定を行います。

OPEN "b:name.dat" FOR RANDOM AS #1 LEN=18
FIELD #1,16 AS Name\$, 2 AS Year\$



上の例では、Name\$, Year\$という2つのフィールドからなる18バイトのリード/ライト・バッファを設定しています。GET, PUTは、このバッファとファイルとの間でリード/ライトを行います。

同じファイルに対して、FIELD文で異なるバッファを設定することができ、これらは同時に有効です。

バッファのフィールドとして宣言されている変数(上の例のName\$, Year\$)に対して、INPUTや代入文で値を代入しないでください。

例 CVI, MKD\$参照.

FILEATTR (ファイル属性の取得) 関数

書式 FILEATTR(n, flag)
↑ ↑ 取得情報の種類
ファイル番号

機能 flag=1のときは、ファイル番号nのファイルのモードを次の値で返します。

- 1 INPUT
- 2 OUTPUT
- 4 RANDOM
- 8 APPEND
- 32 BINARY

flag=2のときは、ファイル番号nのファイルのMS-DOS上のファイルハンドルを返します。

例

```
OPEN "LPT1:" FOR OUTPUT AS #1
OPEN "abc.dat" FOR INPUT AS #2

PRINT "ファイル番号", "ハンドル", "モード"
PRINT "#1", FILEATTR(1, 2), FILEATTR(1, 1)
PRINT "#2", FILEATTR(2, 2), FILEATTR(2, 1)
```

ファイル番号	ハンドル	モード
#1	5	2
#2	6	1

FILES (ファイル一覧の表示)

書 式 FILES [path]
 ↑ファイル名またはパス名

機 能 path で指定したファイルまたはサブディレクトリ内のファイルの一覧を表示します。
path を省略すると、カレントディレクトリ内のすべてのファイル一覧が表示されます。
path にワイルドカード(*と?)を使うこともできます。

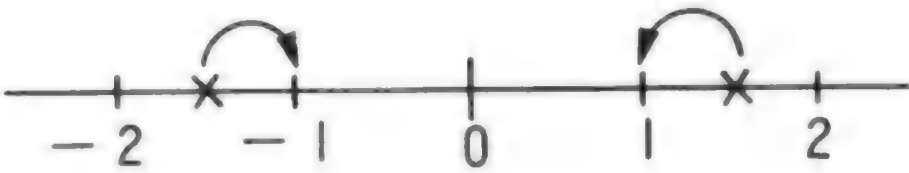
例 FILES "B:* .BAS "
 …ドライブ B のファイルタイプ .BAS を持つすべてのファイルを表示します。

FILES "%BIN%WORK "
 …サブディレクトリ %BIN%WORK のファイルを表示します。

FIX (小数部切り捨て) 関数

書 式 FIX(x)

機 能 実数値 x の小数部を切り捨て、整数部だけを返します。



例 INT 参照.

FOR ~NEXT

(所定回反復)

書 式

```
FOR  var = start TO end [STEP n]
      ↑           ↑           ↑           ↑
      初期値     最終値     きざみ値
      ↓
NEXT [var, ...]
```

ループ変数

機 能

FOR と NEXT で囲まれた範囲を所定の回数だけくり返します。所定の回数とは、初期値から最終値までループを1回くり返すたびに、きざみ値で示された値ずつ増加または減少させていった回数です。

初期値、最終値、きざみ値には、変数、定数、式を書くことができ、その値は正でも、負でも小数であってもかまいません。

きざみ値 n が1のときだけ STEP n を省略できます。

FOR のあとのループ変数と、NEXT のあとのループ変数とが異なってはいけません。ただし、NEXT のあとのループ変数は省略することができます。

FOR ループ内でループ変数の内容を変更したり、GOTO 文で FOR ループ内に入ってはいけません。

FOR ループから強制的に抜けるには EXIT FOR を用います。

例

```
FOR J=1 TO 10
  ⋮
NEXT J
```

----- J を1から始め1きざみで10になるまで、FOR と NEXT で囲まれた部分をくり返す。

```
FOR A=5 TO 0 STEP -0.5
  ⋮
NEXT A
```

----- A を5から始め、-0.5きざみで0になるまでくり返す。A の値は5, 4.5, 4...1, 0.5, 0 と変化する。

```
FOR J=1 TO 5
  FOR K=1 TO 10
    ⋮
  NEXT K
NEXT J
```

----- 2重ループ。外側のループが1回まわるごとに内側のKループは10回まわる。したがって、5×10回くり返しが行われる。

FRE

(使用可能なメモリ・サイズの取得)

関数

書 式

```
FRE(flag)
      ↑
      調べるメモリの種類
```


機能

flag で示されるメモリの使用可能なサイズを取得します。

flag	機能
-1	これから設定できる最大の非文字列配列のサイズをバイト数で返す
-2	スタックスペースの残りバイト数を返す
その他の数値	使用可能な文字列記憶領域のバイト数を返す
文字列	使用可能な文字列記憶領域のバイト数を返す

FRE は使用可能なメモリ・サイズを取得する前に、使用可能な領域を圧縮処理して1つのブロックにします。

例

```
DECLARE SUB abc ()
PRINT TAB(14); "文字列配列", "非文字列配列", "スタック"
PRINT "main 設定前:", FRE(0), FRE(-1), FRE(-2)

' $DYNAMIC
DIM a(100), a$(100)      ' ダイナミック配列の設定

PRINT "main 設定後:", FRE(0), FRE(-1), FRE(-2)

abc

REM $STATIC
SUB abc
  DIM a, b, c      ' ローカル変数
  PRINT "procedure : ", FRE(0), FRE(-1), FRE(-2)
END SUB
```

	文字列配列	非文字列配列	スタック
main 設定前:	46670	173822	1152
main 設定後:	46258	172994	1152
procedure :	46258	172994	1130

FREEFILE (未使用ファイル番号の取得) 関数

書式

FREEFILE

機能

次に使用できるファイル番号を取得します。この関数を使用すれば、プロシージャにおいて、すでに使用しているファイル番号を二重に指定してしまうことはありません。

例

```
’ ---/* 作業用ファイルへのコピー */---  
DECLARE SUB TempFile (n!)
```

```
OPEN "abc.dat" FOR INPUT AS #1
```

```
TempFile (1)
```

```
CLOSE #1
```

```
SUB TempFile (n)
```

```
FileNum = FREEFILE ← 次に使用できるファイル番号の取得
```

```
OPEN "work.tmp" FOR OUTPUT AS FileNum
```

```
WHILE NOT EOF(n)
```

```
  a$ = INPUT$(1, n) ← 1 文字入力
```

```
  PRINT #FileNum, a$; ← 1 文字出力
```

```
WEND
```

```
CLOSE FileNum
```

```
END SUB
```

…プロシージャ TempFile は、work.tmp というファイルをファイル番号 FileNum でオープンし、ファイル番号 n のファイルを 1 文字単位でここにコピーする。

FUNCTION

(関数プロシージャの定義)

書 式

```
FUNCTION name[( var [AS type], ...)] [STATIC]
```

```
  :
```

↑ 仮引数 ↑ 仮引数の型
↑ プロシージャ名

```
  name = expression
```

↑ 関数の戻り値

```
END FUNCTION
```

機 能

name で示される関数プロシージャを定義します。関数の型は、name のあとに型宣言文字(%, &, !, #, \$)を付けるか、DEFtyp 文を用いて名前のインプリシット宣言をして指定します。

() の中に仮引数を書きます。仮引数があれば() を省略できます。

関数の戻り値は、プロシージャ名 name に expression で示す式の値を代入することで得られます。もし name への代入が置かれていなければ、数値型関数の戻り値は 0、文字列型関数の戻り値はヌル文字列となります。

STATIC を指定すると、関数プロシージャ内のローカル変数はすべて(SHARED 宣言されている変数は除く)静的変数となります。静的変数については第 4 章4-2の 8 を参照してください。

例

’ ---/* データ入力関数 */---

DECLARE FUNCTION Dinput! (x!)

Sum = 0
WHILE Dinput(a) <> -1
 Sum = Sum + a
WEND
PRINT "合計="; Sum
END

-----関数プロシージャ Dinput は引数 x に入力データを返し、データ入力の終わりには関数戻り値 -1 を返すものとする。

FUNCTION Dinput (x)
 INPUT "data "; x
 IF x = -9999 THEN
 Dinput = -1 データの終わり
 ELSE
 Dinput = 1
 END IF
END FUNCTION

data ? 1
data ? 2
data ? 3
data ? 4
data ? 5
data ? -9999
合計 = 15

GET (ランダム・ファイルからのリード)

書 式

- ① GET [#]n [,rec]
 ↑ ファイル ↑
 ↓ ル番号 ↓ レコード番号
- ② GET [#]n [,rec][,var]
 ↑ リード・データを格納する変数

機 能

- ①ファイル番号 n のファイルから、FIELD 文で設定されているリード/ライト・バッファにデータをリードします。
- ②ファイル番号 n のファイルから、変数 var にデータをリードします。var を指定した場合は、FIELD 文でリード/ライト・バッファを設定しないでください。

rec はランダム・ファイルでは読み出すレコード番号、バイナリ・ファイルでは読み出しを始めるバイト位置です。レコード番号、バイト位置の先頭番号は1です。

rec を省略すると、ファイル現在位置(最後に GET/PUT でアクセスしたレコードの次のデータ、または SEEK で指定したデータ位置)のデータをリードします。rec は 1 ～ 2³¹ - 1 (2147483647) の範囲の値を指定します。

例

```

OPEN  "b: name.dat " FOR RANDOM AS #1 LEN=18
FIELD  #1, 16 AS Name$, 2 AS Year$
      :
INPUT  Rec
GET    #1, Rec  ←レコード番号 Rec のデータを FIELD で設定したバッファにリード

TYPE  Man
      Namae AS  STRING *16
      Year   AS  INTEGER
END  TYPE
DIM   A AS Man
OPEN  "b: name.dat " FOR RANDOM AS #1 LEN=LEN(A)
      :
INPUT  Rec
GET    #1, Rec,A  ←レコード番号 Rec のデータをレコード型変数 A にリード

```

GET

（画面イメージの保存）

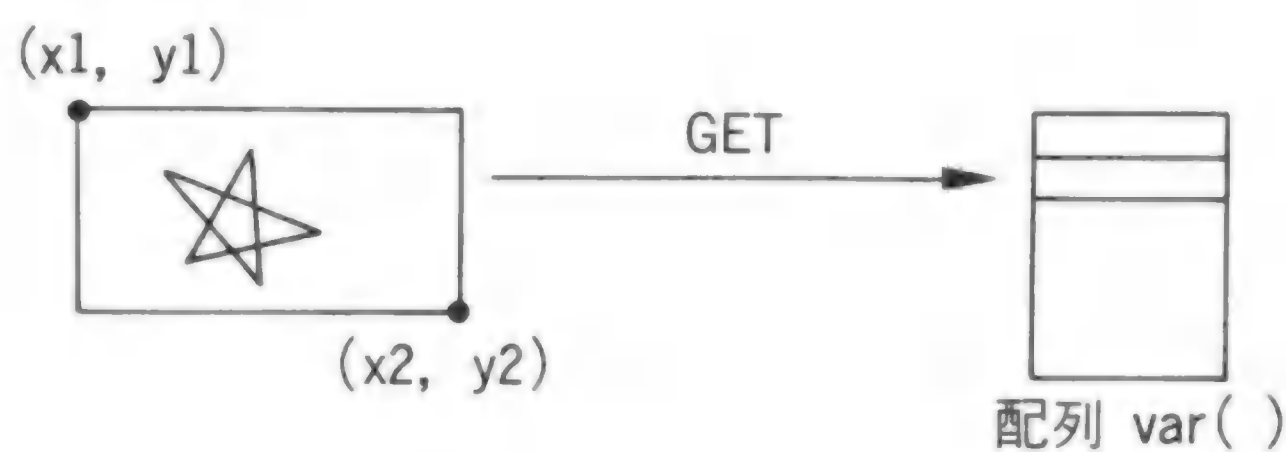
書式

```
GET [STEP](x1,y1) - [STEP](x2,y2), var[(n)]
```



機能

(x1,y1) – (x2,y2) で示される四角形のグラフィック画面のデータを配列 var に保存します.



(x1,y1) - (x2,y2) の画面データを保存するために必要なバイト数 B は,

$$B = 4 + 4 * \text{INT}(((x_2 - x_1 + 1) + 7) / 8) * (y_2 - y_1 + 1)$$

で求められます。したがって必要な配列のサイズ A は、配列のデータ型によって次のように求められます。

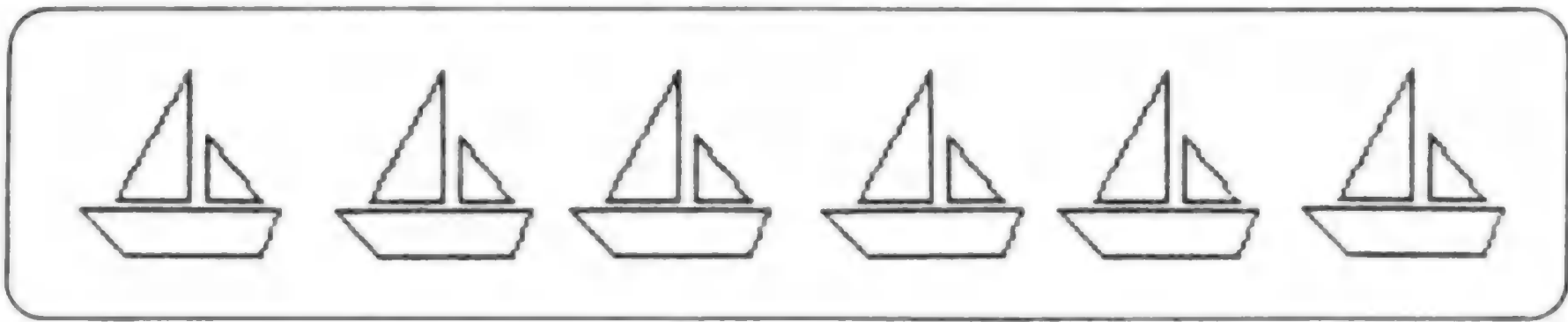
A = INT(B/M) + 1

- 2 : 整数型
- 4 : 倍長整数型
- 4 : 単精度実数型
- 8 : 倍精度実数型

STEP は相対座標を意味し、STEP (x1,y1)は現在位置からの相対距離、STEP (x2,y2)は (x1,y1)からの相対距離となります。
n は、イメージを保存する配列の格納開始要素を示す値です。省略すると配列先頭からになります。

例

```
· ---/* 画面イメージのGET/PUT */---  
  
SCREEN 0: CLS  
x = 45: y = 41 ← 領域の横・縦のドット数  
byte = 4 + 4 * ((x + 7) \ 8) * y · 必要なバイト数  
s = byte \ 2 + 1 · 必要な配列要素の数  
DIM p%(s) · イメージ用配列  
  
COLOR 1  
LINE (0, 30)-(10, 40): LINE -(40, 40)  
LINE -(44, 30): LINE -(0, 30) ← ヨットの描画  
LINE (24, 0)-(8, 28): LINE -(24, 28): LINE -(24, 0)  
LINE (28, 14)-(28, 28): LINE -(40, 28): LINE -(28, 14)  
  
GET (0, 0)-(44, 40), p% ← ヨット・パターンの取得  
  
CLS  
FOR x = 0 TO 300 STEP 60 ] ← ヨットを横に表示していく  
    PUT (x, 0), p%, PSET  
NEXT x
```



GOSUB (サブルーチンの呼び出し)

書 式 GOSUB line
 ↑ 飛び先

機 能 line で示されるサブルーチンを呼び出します。line にはラベルまたは行番号を指定します。
サブルーチンからは RETURN で戻ります。

例

```

GOSUB      ABC
:
ABC:
  サブルーチン本体
RETURN

```

...ラベル ABC で示されるサブルーチンへ分岐する。サブルーチンの RETURNにより、GOSUB の次の文に戻る。

```

GOSUB      1000
:
1000  サブルーチン本体
RETURN

```

...1000行のサブルーチンへ分岐する。

GOTO

(分岐)

書 式

GOTO line
 ↑ 飛び先

機 能

line で示される位置へ分岐します。line にはラベルまたは行番号を指定します。

例

```

GOTO      OWARI
:
OWARI:

```

...ラベル OWRI で示される位置へ分岐する。

HEX\$

(数値を16進文字列に変換)

関数

書 式

HEX\$(n)
 ↑ 数値

機 能

数値 n を16進文字列に変換して返します。

例

A\$ = HEX\$(255) ...FF が返される。

IF THEN ELSE

(条件判断)

書 式

① IF p THEN s1 [ELSE s2]

↑ ↑ ↑
条件式 文 文

↓ 条件式

② IF p1 THEN
 s1 ← 文
 [ELSEIF p2 THEN
 s2]
 ⋮
 [ELSE
 sn]
END IF

機 能

①単純 IF. 条件 p を満たせば s1を, 満たさなければ s2を実行します. s1,s2にはコロン(:)で区切ってマルチ・ステートメントを書くことができます. 単純 IF 文は1行に収まる範囲です.

②ブロック IF. 条件 p1を満たせばブロック s1を, そうでなく p2を満たせばブロック s2を, …, いずれも満たさなければブロック snを実行します.

ELSEIF 節, ELSE 節は省略できますが, END IF は省略できません.

ブロック s1~sn は複数行からなる文の集まりです.

例

IF A>0 THEN SP = SP + A ELSE SM = SM + A

…A>0 なら SP = SP + A を, そうでなければ SM = SM + A を行う.

IF A>0 THEN

SP = SP + A
PRINT SP

← ①

ELSE

SM = SM + A
PRINT SM

← ②

END IF

…A>0ならブロック①を, そうでないなら
ブロック②を実行する.

```

IF A>80 THEN
    

ブロック A


ELSEIF A>70 THEN
    

ブロック B


ELSEIF A>60 THEN
    

ブロック C


ELSE
    

ブロック D


END IF

```

… A>80ならブロック A を,
 そうでなく A>70ならブロック B を,
 そうでなく A>60ならブロック C を,
 それら以外ならブロック D を実行する.

INKEY\$

(待ったなしのキー入力)

関数

書式

INKEY\$

機能

入力されたキー文字を返します。キー入力がされていなければ、ヌル文字列を返します。
 キーから読んだコードが漢字コードまたは拡張コードの場合、2バイト文字を返します。
 INKEY\$は入力文字を画面上にエコーバックしません。
 次のキーはINKEY\$では読み込むことはできません。

CTRL

+

C

プログラムの終了

COPY

画面のコピー

ただし、コンパイル時に D/デバッグコードまたは /d オプションを指定していない独立型の .EXE プログラムは、

CTRL

 +

C

 を読むことができます。

例

```

CLS
DO
    a$ = INKEY$
    IF a$ <> "" THEN PRINT ">";
LOOP WHILE a$ <> CHR$(27)

```

…キーを押すたびに“>”を表示する。

ESC

でループから抜ける。

INP

(ポートから1バイト入力)

関数

書式

INP(n)
↑ポート番号

機能

n番(0~65535)のポートから1バイトのデータを入力します。

例

Lputc は、プリンタポートに直接データを出力する関数です。
Lputs は、Lputc を用いて文字列をプリンタに出力します。

```
DECLARE SUB Lputc (c%)
DECLARE SUB Lputs (a$)

Lputs ("Hello Quick")
Lputc (&HD): Lputc (&HA)

SUB Lputc (c%)      ' 直接プリンタ出力
  WHILE (INP(&H42) AND 4) <> 4
    WEND
  OUT &H40, c%
  OUT &H46, 14
  OUT &H46, 15
END SUB

SUB Lputs (a$)      ' 直接文字列出力
  FOR k = 1 TO LEN(a$)
    c% = ASC(MID$(a$, k, 1))
    Lputc c%
  NEXT k
END SUB
```

←第2ビットがセット(1)されるまで待つ

INPUT\$

(指定したバイト数分の文字列のリード)

関数

書式

INPUT\$(size[, #n])
↑↑ファイル番号
リードバイト数

機能

ファイル番号 n のファイルから size バイトの文字をリードします。
ファイル番号 n を省略すると、標準入力(通常キーボード)からリードします。

例

次のプログラムは、ファイルから1バイト単位でリードし、スクリーンに表示していくものです。
MS-DOS のテキスト・ファイルの行末は CR・LF(&H0D,&H0A) の2バイトで構成され、QB でこの文字をスクリーンに2回表示すると改行が2度行われるため、最初の&H0D は出力しないようにしました。

```

' ---/* ファイルのタイプ */---

CLS
INPUT "FileName "; File$
OPEN File$ FOR INPUT AS #1

DO WHILE NOT EOF(1)
    a$ = INPUT$(1, #1) ← 文字(バイト)リード
    IF a$ <> CHR$(&HA) THEN PRINT a$;
LOOP
CLOSE #1

```

INPUT¥

(指定した長さの文字のリード)

関数

書 式

INPUT¥(size[, #n])

↑ ↑
リードする文字数 ファイル番号

機 能

ファイル番号 n のファイルから、size で示される個数の文字をリードします。1 バイト文字なら 1 バイトを、2 バイト文字なら 2 バイトを 1 文字として扱います。INPUT\$ はバイト単位のリードです。

例

次のプログラムは、ファイルから 1 文字単位でリードし、スクリーンに表示して行くものです。INPUT\$ で示した例と同じものですが、こちらは、2 バイト文字を含むファイルに対しても正常に動作します。

```

' ---/* ファイルのタイプ */---

CLS
INPUT "FileName "; File$
OPEN File$ FOR INPUT AS #1

DO WHILE NOT EOF(1)
    a$ = INPUT¥(1, #1) ← 文字リード
    IF a$ <> CHR$(&HA) THEN PRINT a$;
LOOP
CLOSE #1

```

INPUT

(キーボードからのデータ入力)

書 式

INPUT [;] ["msg" {; | , }] var, ...

↑ ↑
メッセージ文字列 変数

機能

変数 var にキーボードからデータを入力します。
INPUT 直後にセミコロン(;)を置くと、データ入力時に ☐ を押してもカーソルは同じ行にとどまります。
msg はメッセージ文字列で、その後に置く“;”と“,”は機能が異なります。

- ; ... msg のあとに?を表示する。
- , ... msg のあとに?を表示しない。

例

INPUT "data" ; A

data ? 20 ☐

INPUT "data" ; X, A\$

data ? 20, Hello ☐

INPUT "", A

20 ☐

INPUT # (シーケンシャル・ファイルからのリード)

書式

INPUT # n, var, ...

↑ ↑
ファイル番号 変数

機能

ファイル番号 n のシーケンシャル・ファイルから、データを変数 var にリードします。

例

```
' ---/* シーケンシャル・ファイルの読み込み */---  
  
OPEN "man.dat" FOR INPUT AS #1  
  
PRINT "Name", "Year"  
DO WHILE NOT EOF(1)  
    INPUT #1, Name$, Year  
    PRINT Name$, Year  
LOOP  
CLOSE #1
```

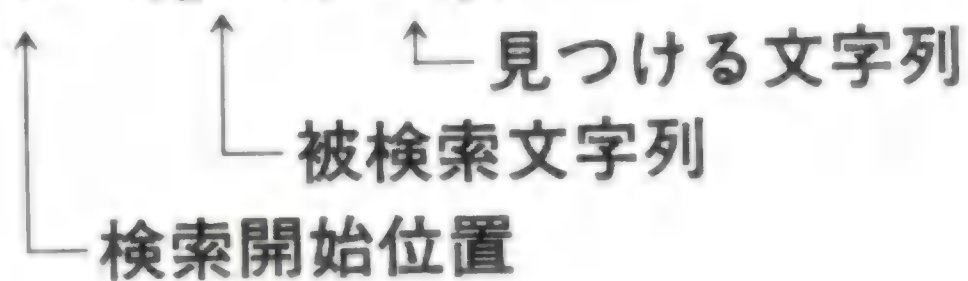
INSTR

(文字列の検索)

関数

書 式

INSTR ([start,]str, key)



機 能

文字列 str 中から key で示される文字列を探し、その位置(バイト位置)を返します。start は検索開始位置で、省略すると先頭から検索を開始します。

検索文字列が見つからなければ 0 を返します。

例

INSTR("Quick _ BASIC", "BAS")... 7 を返します。
次のプログラムは文字列中の空白の位置を表示するものです。

```
Text$ = "THis is a pen"  
p = INSTR(Text$, " ")  
DO WHILE p <> 0  
    PRINT p  
    p = INSTR(p + 1, Text$, " ")  
LOOP
```

↑ 空白の次の位置を開始点にする

5
8
10

INT

(小数部切り捨て)

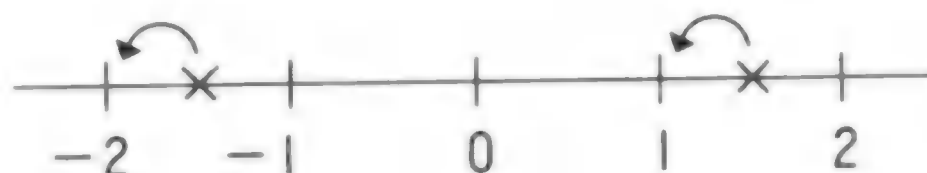
関数

書 式

INT(x)

機 能

実数値 x の小数部を切り捨て、それよりも小さい整数を返します。



例

```
a$ = "ABC日本語"  
FOR k = 1 TO KLEN(a$)  
  b$ = KMID$(a$, k, 1)  
  PRINT b$; "="; JIS$(b$)  
NEXT k
```

A=65
B=66
C=67
日=467C
本=4B5C
語=386C

KEXT\$ (1バイト文字または2バイト文字の抽出) 関数

書式

KEXT\$ (str, func)

↑ 抽出する文字の種類
↑ 文字列

機能

func=0のときは、str から1バイト文字だけを抜き出して返し、
func=1のときは、str から2バイト文字だけを抜き出して返します。
指定した文字列が存在しなければ、ヌル文字列が返されます。

例

```
a$ = "日本語ABC入門123"  
PRINT KEXT$(a$, 0), KEXT$(a$, 1)
```

ABC123	日本語入門
--------	-------

KEY (ファンクション・キー制御)

書式

- ① KEY n, str
- ↑ ファンクション・キーに割り当てる文字列
↑ ファンクション・キーの番号(1~10)
- ② KEY LIST
③ KEY ON
④ KEY OFF

機能

- ① n(1～10)で示されるファンクション・キーに文字列を割り当てます。文字列は最大15文字までで、それを越えた文字は捨てられます。
- ② ファンクション・キーに割り当てられている文字列の一覧を表示します。
- ③ 画面の最下行に各ファンクション・キーに割り当てられている最初の6文字を表示します。
- ④ 画面の最下行へのファンクション・キーの表示を行いません。

KEYにより設定したファンクション・キーが機能するのは、プログラムの実行中で、QB環境下ではありません。

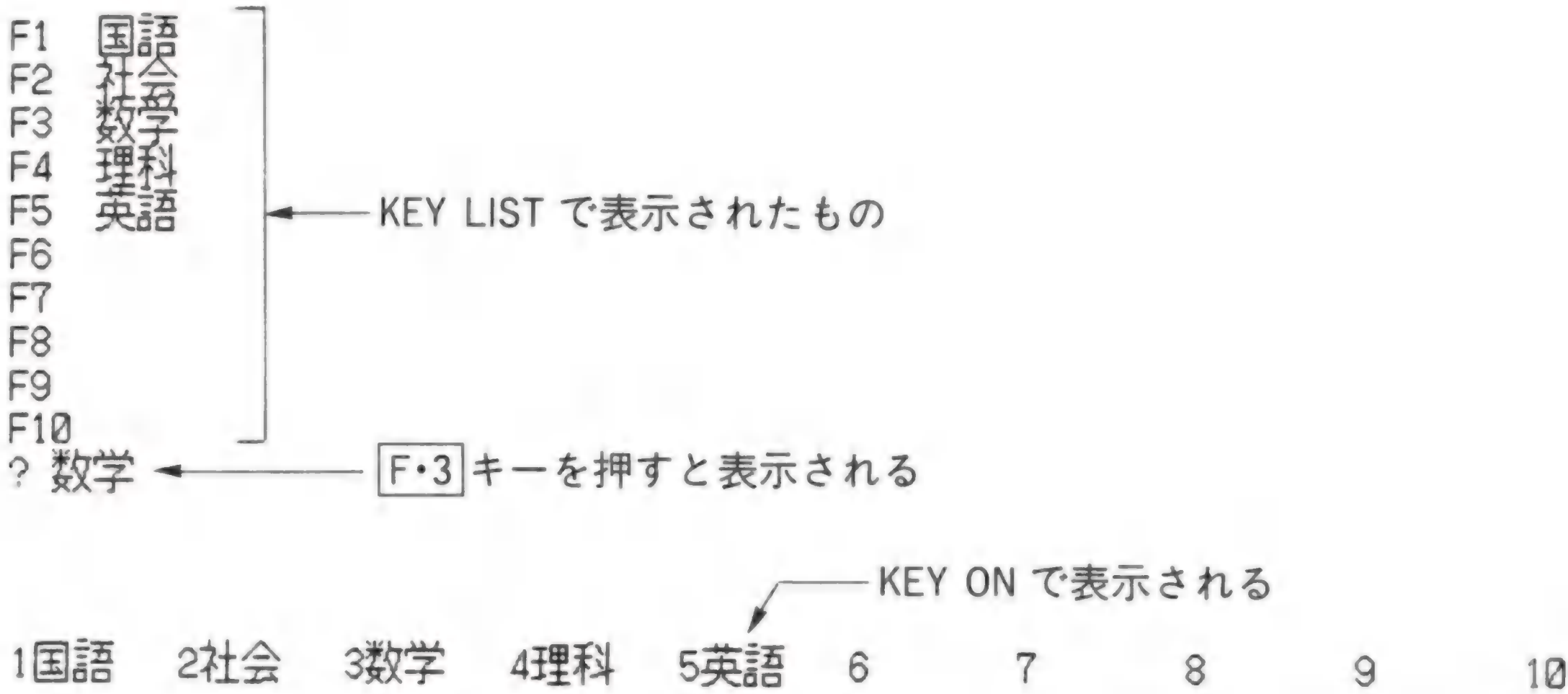
例

```
CLS
FOR k = 1 TO 5
  READ a$
  KEY k, a$
NEXT K
DATA 国語, 社会, 数学, 理科, 英語

KEY ON
KEY LIST

INPUT Text$
```

← F・1～F・5にデータを割り当てる



KEY(n) (キー・トラッピングの開始／禁止／停止)

書 式

- ① KEY(n) ON
 - ② KEY(n) OFF
 - ③ KEY(n) STOP
- ↑ キー番号

機 能

- ①指定したキーのトラッピングを開始
 - ②指定したキーのトラッピングを禁止
 - ③指定したキーのトラッピングを停止. 停止中に発生したトラッピングは記憶されており, トラッピングが許可(開始)されるとただちにイベント処理ルーチンに分岐します.
- キー番号 n とキーの関係は次のとおりです.

n	キ ー
1~10	F・1 ~ F・10
11	↑
12	←
13	→
14	↓
15~25	ユーザ定義キー

ユーザ定義キーは KEY 文を用いて次のように定義します.

KEY n, CHR\$(keyflag)+CHR\$(scancode)

keyflag の値は以下のとおりです.

keyflag	キ ー
&H00	キーボードフラグなし
&H01	SHIFT
&H02	CAPS
&H04	カナ
&H08	GRPH
&H10	CTRL

scancode は次ページの表のとおりです.

たとえば,

KEY 15, CHR\$(&H10)+CHR\$(&H1E)

は, CTRL + S を15番のキーに割り当て,

KEY 15, CHR\$(&H00)+CHR\$(&H1E)

は, S を15番のキーに割り当てます.

● scancode

キー	コード	キー	コード	キー	コード	キー	コード
ESC	00	P	19	/ ?	32	2	4B
1 !	01	@ ~	1A	—	33	3	4C
2 "	02	[{	1B	スペース	34	=	4D
3 #	03	改行	1C	XFER	35	0	4E
4 \$	04	A	1D	ROLL UP	36	,	4F
5 %	05	S	1E	ROLL DOWN	37	.	50
6 &	06	D	1F	INS	38	NFER	51
7 '	07	F	20	DEL	39	STOP	60
8 (08	G	21	↑	3A	COPY	61
9)	09	H	22	←	3B	f•1	62
0	0A	J	23	→	3C	f•2	63
- =	0B	K	24	↓	3D	f•3	64
^ _	0C	L	25	HOMECLR	3E	f•4	65
¥	0D	; +	26	HELP	3F	f•5	66
BS	0E	: *	27	—	40	f•6	67
TAB	0F] }	28	/	41	f•7	68
Q	10	Z	29	7	42	f•8	69
W	11	X	2A	8	43	f•9	6A
E	12	C	2B	9	44	f•10	6B
R	13	V	2C	*	45	SHIFT	70
T	14	B	2D	4	46	CAPS	71
Y	15	N	2E	5	47	カナ	72
U	16	M	2F	6	48	GRPH	73
I	17	, <	30	+	49	CTRL	74
O	18	• >	31	1	4A		

例

KEY 15, CHR\$(&H10) + CHR\$(&H1E) 'ユーザ定義キー ← CTRL + S をキー番号15に定義

```
KEY(1)ON 'トラップの開始
KEY(15)ON
ON KEY(1) GOSUB Key1 'トラップ・ルーチンの設定
ON KEY(15) GOSUB Key2
```

```
n = 1: m = 1
WHILE 1 '無限ループ
WEND
```

```
Key1:
PRINT m: ":now trap key1"
m = m + 1
RETURN
```

```
Key15:
PRINT n: ":now trap key15"
n = n + 1
RETURN
```

1 :now trap key1
1 :now trap key15
2 :now trap key1
3 :now trap key1

KLEN (文字列長さの取得: LEN の漢字対応版) 関数

書 式 KLEN(str)
 ↑文字列

機 能 文字列の長さを文字数で返します。1 バイト文字、2 バイト文字とも 1 文字として数えます。

例

a\$ = "日本語ABC"

```
n = LEN(a$)
```

```
m = KLEN(a$)
```

```
PRINT "バイト数" =": n
```

PRINT "文字数" = " : 11

```
PRINT "2 バイト文字の数="; n - m
```

バイト数 = 9
文字数 = 6
2バイト文字の数 = 3

KMID\$ (中間文字列の取得：MID\$の漢字対応版) 関数

書式

```
KMID$(str, start[, n])
```

- 文字列
- 取り出す開始位置
- 取り出す文字数

機能 文字列 str の start 位置から、n 個の文字を取り出して返します。1 バイト文字、2 バイト文字とも 1 文字として数えます。n を省略すると、文字列の終わりまで取り出します。

例

a\$ = “日本語abc”

```
FOR k = 1 TO KLEN(a$)
  PRINT KMIDS$(a$, k, 1)
NEXT k
```

日本語
a
b
c

KMID\$ (文字列の置き換え：MID\$の漢字対応版)

書 式

KMID\$(strvar, start[, n])= strexpression

↑ 置き換える文字列
↑ 置き換える文字数
↑ 置き換え開始位置
↑ 置き換えられる文字列

機 能

文字列変数 strvar の start 位置以後の文字を、strexpression で置き換えます。n を指定すると、strexpression の先頭から n 文字分を置き換えます。

例

```
Text$ = "これは船"
Key$ = "船"
Rep$ = "電車"
KMID$(Text$, KINSTR(Text$, Key$)) = Rep$
PRINT Text$
```

…Text\$中から Key\$の文字列を探し、Rep\$で置き換える。船の代わりに電車が入るのではなく、船の代わりに電車が入る。

これは電車

KPOS (指定文字位置までのバイト数の取得) 関数

書 式

KPOS(str, n)

↑ 文字位置
↑ 文字列

機 能

文字列 str の n 番目の文字(1 バイト文字, 2 バイト文字を 1 文字と数える)が、先頭から何バイト目になるかを求めて返します。

例

KPOS("日本語 abc", 5) … 8を返す。
↑ 5番目の文字

KTN\$ (文字を句点コードまたは ASCII コードに変換) 関数

書 式

KTN\$(str)

↑ 文字列

機 能

文字列 str の先頭 1 文字が 2 バイト文字のときは、それに対応する句点コードを、1 バイト文字のときは、それに対応する ASCII コードを文字列の形で返します。

例

```
a$ = "O A ア亜日本語"

PRINT TAB(14); "句点コード", "J I Sコード"
FOR k = 1 TO KLEN(a$)
    b$ = KMID$(a$, k, 1)
    PRINT b$, KTN$(b$), JIS$(b$)
NEXT k
```

	句点コード	J I Sコード
O	0316	2330
A	0333	2341
ア	0502	2522
亜	1601	3021
日	3892	467C
本	4360	4B5C
語	2476	386C

LBOUND

(配列添字の下限値の取得)

関数

書式

LBOUND(array[, d])

↑

添字の次元

↑

配列名

機能

配列 array の d 次の添字の下限値を取得します。

LBOUND, UBOUND は、引数渡しされた、配列のサイズを取得するのに使います。

```
DIM A(-10 TO 20, 5 TO 10)
```

LBOUND(A, 1) ... -10 を返す。

LBOUND(A, 2) ... 5 を返す。

例

```
プロシージャ SetArray は 2 次元配列を受けとり、各要素を dat で初期化します。

DECLARE SUB SetArray (b!(), dat!)
DIM a(-2 TO 2, 3 TO 5)

SetArray a(), -1

SUB SetArray (b(), dat)
    FOR j = LBOUND(b, 1) TO UBOUND(b, 1) ← 第 1 次の添字の下限と上限
        FOR k = LBOUND(b, 2) TO UBOUND(b, 2) ← 第 2 次の添字の下限と上限
            b(j, k) = dat
        NEXT k
    NEXT j
END SUB
```

LCASE\$

(小文字に変換)

関数

書 式

LCASE\$(str)
↑ 文字列

機 能

文字列中の大文字を小文字に変換して返します。対象となる文字は1バイト文字、2バイト文字のA~Zです。

例

```
INPUT A$
IF LCASE$(A$) = "yes" THEN~
```

…A\$に Yes, YES, などが入力されても yes として比較できる。

LEFT\$

(左部分文字列の取得)

関数

書 式

LEFT\$(str, n)
↑ 左端からの文字数
↑ 文字列

機 能

文字列 str の左端から n 文字(バイト)を取り出します。n が文字列の長さより大きければ文字列全体を返し、n が 0 ならヌル文字列を返します。

例

```
A$ = "Hello BASIC"
B$ = LEFT$(A$, 5) … Hello が返される。
```

LEN

(文字列長さの取得)

関数

書 式

LEN(str)
↑ 文字列

機 能

文字列 str の長さを返します。2バイト文字は2文字と数えます。日本語を含む文字列には KLEN を使用してください。

例

```
LEN("Hello") … 5 を返す。
LEN("日本語 ABC")… 9 を返す。
```


LEN

(文字のサイズの取得)

関数

書式

LEN(var)
 ↑ 変数名

機能

変数がメモリ上で使用するバイト数を取得します。

例

TYPE Man
 Name AS STRING * 16
 Year AS INTEGER
END TYPE
DIM A AS Man

LEN(A) ... 16+2=18バイトが返される。

LET

(代入)

書式

[LET] var = expression
 ↑ 変数 ↑ 式

機能

変数に式を代入するための文です。一般に LET は省略し、
 var = expression
だけで代入文として扱われます。

LINE

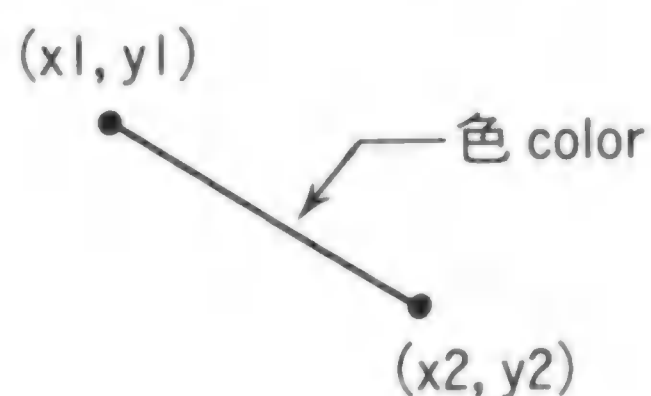
(直線の描画)

書式

LINE [[STEP](x1, y1)] - [STEP](x2, y2), [color],[{B | BF}][,style]
 ↑ 始点 ↑ 終点 ↑ 色 ↑ Box, Box Fill ↑ ラインスタイル

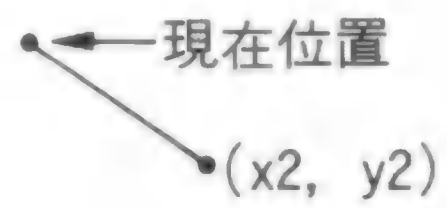
機能

①通常 of 直線
LINE (x1, y1) - (x2, y2) [,color]
色 color を省略すると COLOR 文で設定されている色で描きます。



②現在位置からの描画

LINE -(x2, y2)[,color]



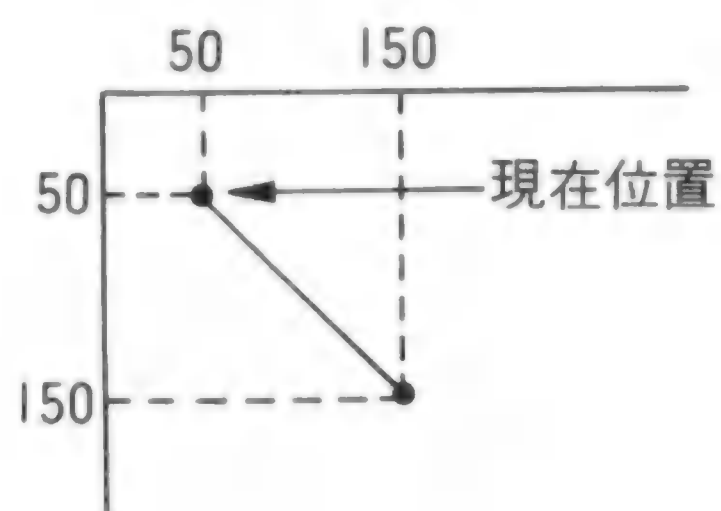
始点座標(x1, y1)を省略すると、現在位置からの描画となります。

③相対指定

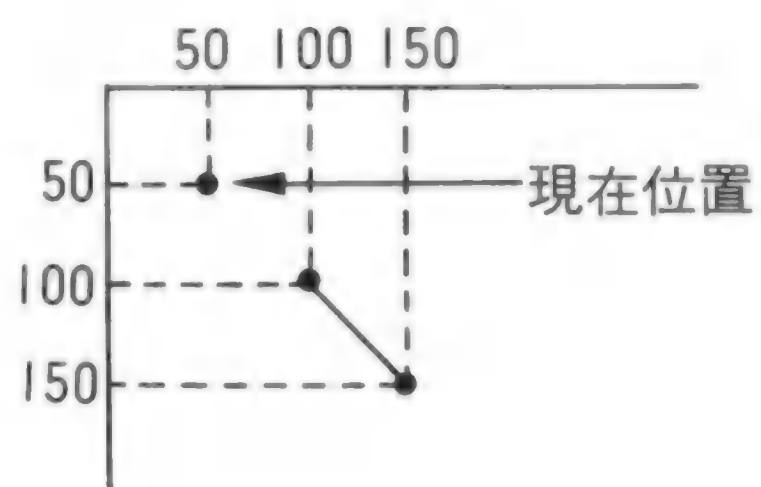
LINE [[STEP](x1, y1)]-[STEP](x2, y2)[,color]

第1座標にSTEPを付けると、現在位置からの距離、第2座標にSTEPを付けると、第1座標からの距離となります。

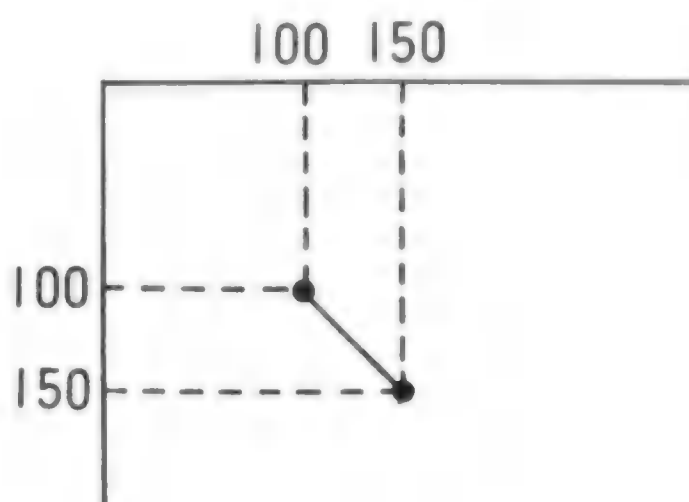
LINE -STEP(100, 100)



LINE STEP(50, 50)-STEP(50, 50)



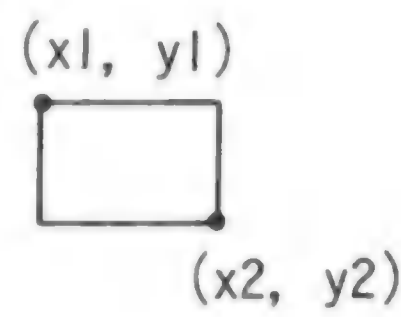
LINE (100, 100)-STEP(50, 50)



④ B(Box), BF(Box Fill)

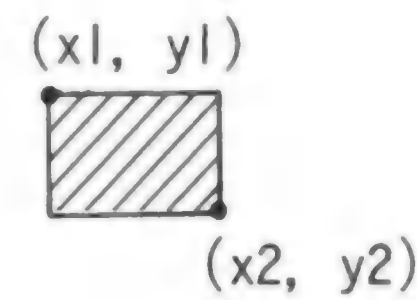
B 指定すると (x1, y1)-(x2, y2) を対角とする四角形を描きます。座標に STEP を指定することもできます。

```
LINE (x1, y1)-(x2, y2),[color], B
```



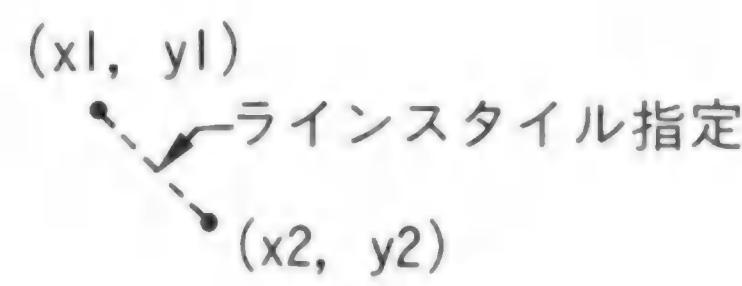
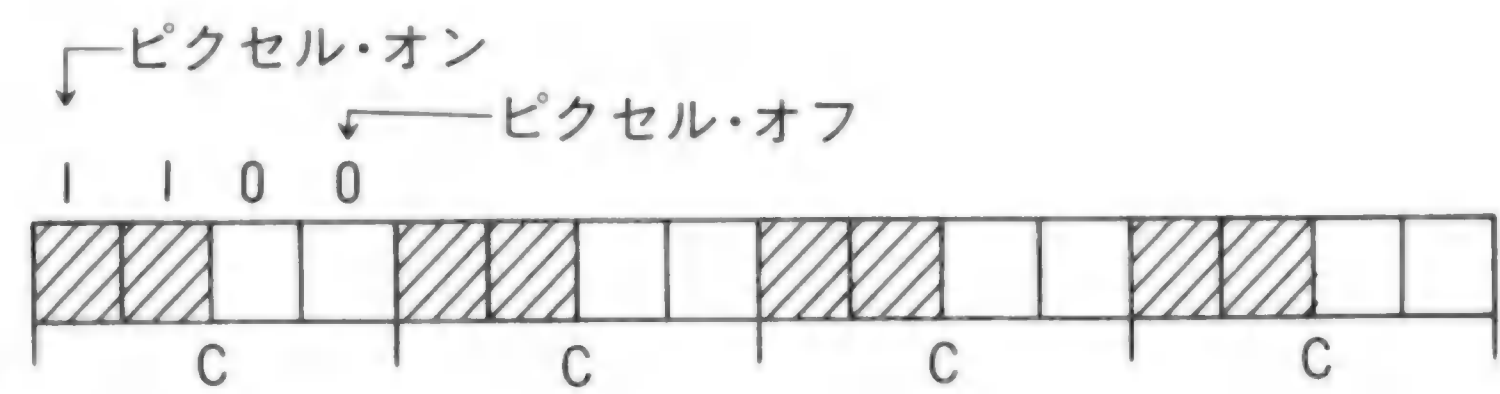
BF 指定すると (x1, y1)-(x2, y2) を対角とする四角形を描き，その中をぬりつぶします。座標に STEP を指定することもできます。

```
LINE (x1, y1)-(x2, y2),[color], BF
```



⑤ ラインスタイルの指定

style に次のような16ビットのビットパターン値を与えることで，線のスタイル(点線，破線など)を指定できます。



```
LINE (x1, y1) - (x2, y2),[color],, &HCCCC
```

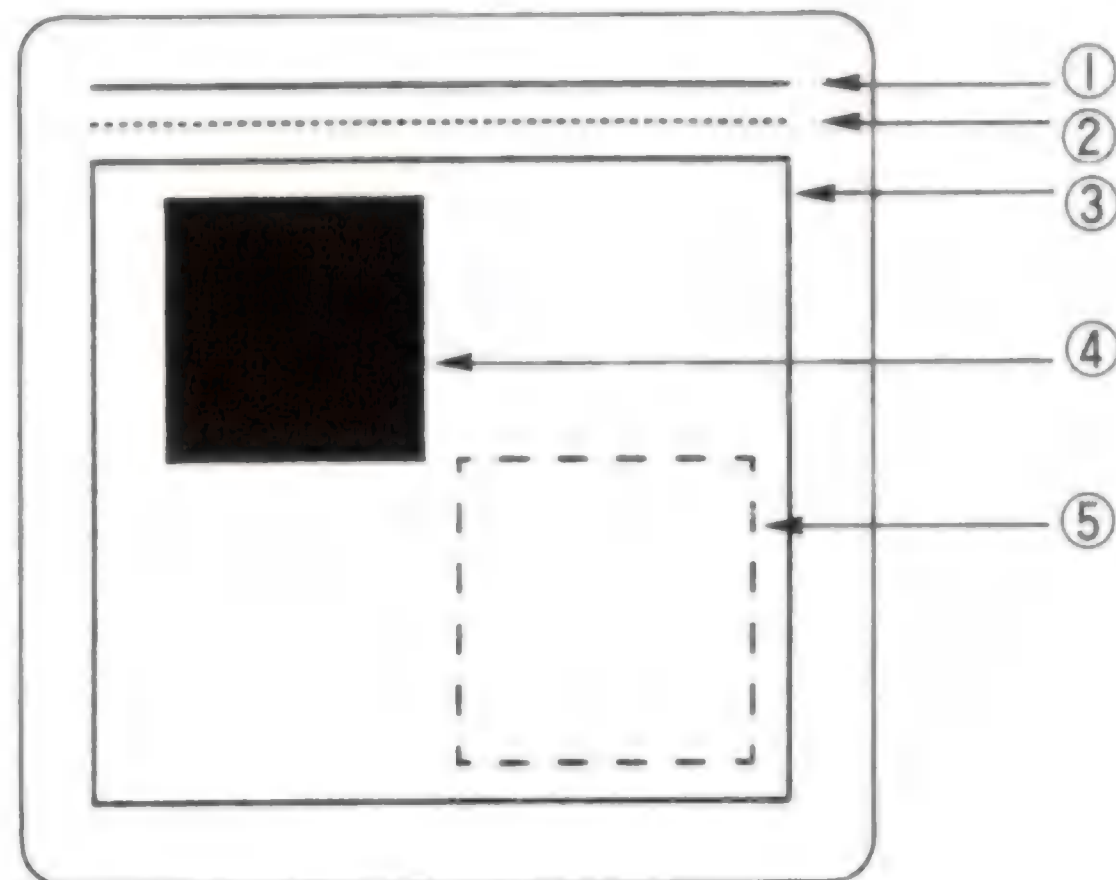
ラインスタイルの指定は BF オプションに対しては機能しません。

例

---/* 直線の描画 */---

SCREEN 0: CLS

```
LINE (10, 10)-(200, 10), 1 ← ①
LINE (10, 20)-(200, 20), 2, , &HCCCC ← ②
LINE (10, 30)-(200, 200), 3, B ← ③
LINE (30, 40)-(100, 110), 4, BF ← ④
LINE (110, 110)-(190, 190), 5, B, &HF00F ← ⑤
```



LINE INPUT

(キーボードから1行入力)

書式

LINE INPUT [;][*msg*;] strvar
 ↑ ↑
 メッセージ 文字列変数

機能

キーボードから入力された1行のデータを strvar に格納します。データ中にカンマ、スペース、二重引用符などを含めることができます。☒のみ入力するとヌル文字列を返します。

msg は表示されるメッセージです。INPUT 文と異なり、msg 中に ? を含ませないかぎり、 ? を表示しません。

INPUT 直後に“;”を置くと、☒を押したあともカーソルは同じ行にとどまります。

例

キーボードから入力した1行をファイルに書き込みます。

入力の終わりは **CTRL** + **Z** ☒ とします。 **CTRL** + **Z** は &H1A というコードです。

```
OPEN "text.dat" FOR OUTPUT AS #1

LINE INPUT a$
DO WHILE MID$(a$, 1, 1) <> CHR$(&H1A) ← 入力行の先頭文字の判定
  PRINT #1, a$
  LINE INPUT a$
LOOP
CLOSE #1
```


LINE INPUT # (シーケンシャル・ファイルから1行単位で入力)

書式

LINE INPUT #n, strvar

↑ ↑
ファイル番号 文字列変数

機 能	ファイル番号 n のシーケンシャル・ファイルのファイル現在位置から 1 行をリードし、strvar に格納します。1 行の終わりは CR・LF コードで判定されます。
------------	---

例 1行単位でファイルからリードし、行番号を付けて表示するものです。このリストはこのプログラム自身で出力したものです。

```

1: ' ---/* 行番号付きファイル・タイプ */---
2:
3: INPUT "FileName"; f$
4: OPEN f$ FOR INPUT AS #1
5:
6: n = 1
7: LINE INPUT #1, a$      ← 1行リード
8: DO WHILE NOT EOF(1)
9:     PRINT USING "###: "; n;  ← 行番号の表示
10:    PRINT a$
11:    n = n + 1      ← 行番号を1つ進める
12:    LINE INPUT #1, a$
13: LOOP
14: CLOSE #1

```

LOC

(ファイル現在位置の取得)

関数

書式 LOC(n)
 ↑
 ファイル番号

機能 ファンダム・ファイルでは、最後にリード/ライトしたレコード番号を返します。

シーケンシャル・ファイルでは、現在位置のバイト数を128で割った値を返します。

バイナリ・ファイルでは、最後にリード/ライトしたバイト位置を返します。

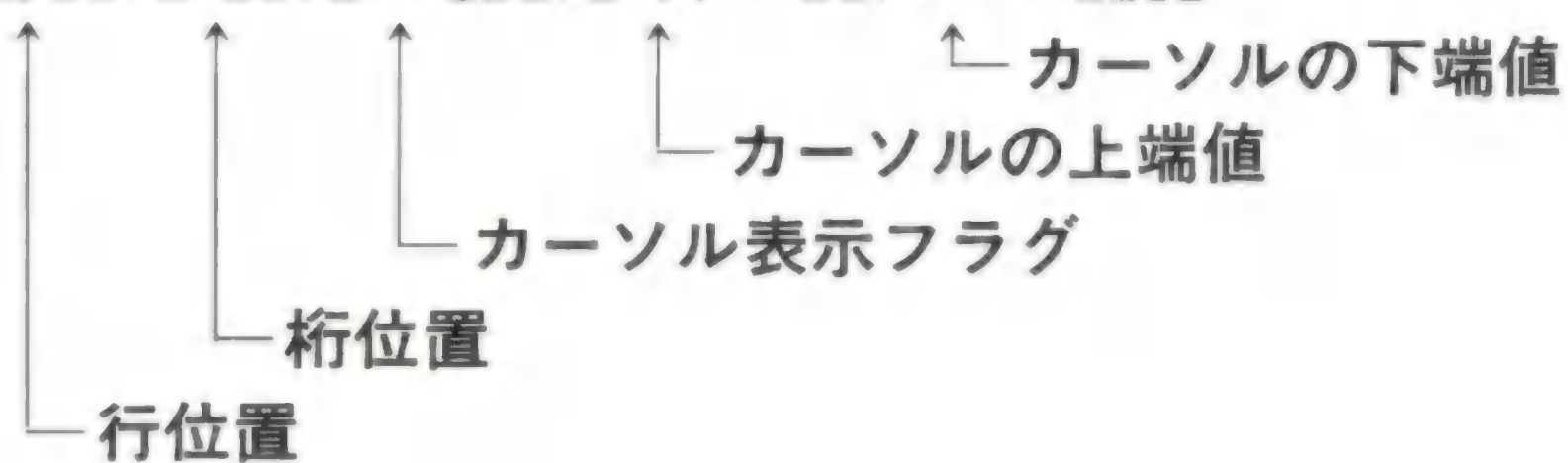
COM(通信)デバイスでは入力バッファにある文字数を返します。

LOCATE

(カーソルを指定位置に移す)

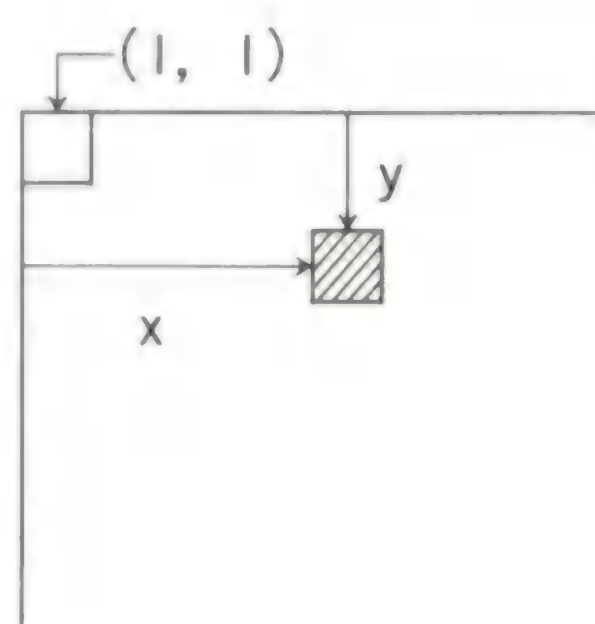
書 式

LOCATE [y][,][x][,][flag][,][upper][,][lower]]]



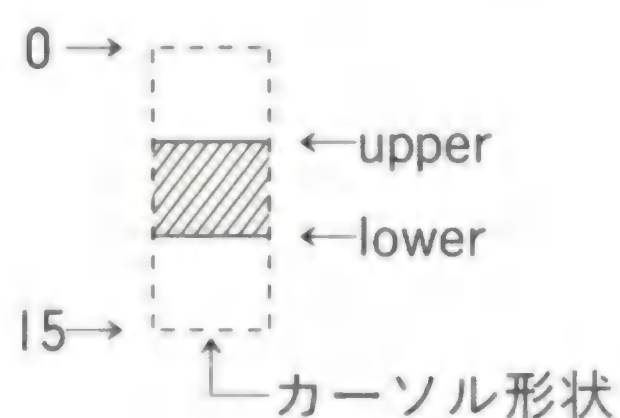
機 能

カーソルを(y, x)で示される位置に移します。y または x を省略すると現在の行または桁位置が使われます。



flag=0とすると、カーソルを表示しません。flag=1とすると、カーソルを表示します。省略すると前の状態を引き継ぎます。

upper と lower はカーソル形状の上端値と下端値です。lower を省略すると、upper と同じ値が採用されます。



例

LOCATE ,, 0 ... カーソルを消す。
 LOCATE ,,, 8, 15 ... カーソル形状を下半分(■)にする。
 LOCATE 2, 10: PRINT "ABC"

	1	2								10			
1													
2										A	B	C	

LOCK/UNLOCK
(他のプロセスからのアクセスの禁止/解除)

書式

LOCK [#]n[, [start][TO end]]

↑ 終末レコード
↑ 先頭レコード
— ファイル番号

```
UNLOCK [#]n[, [start][TO end]]
```

機能

LOCK はファイルの start~end で示される範囲を他のプロセスに対しアクセス禁止にします。start, end はランダム・ファイルではレコード番号、バイナリ・ファイルでは先頭からのバイト位置です。

UNLOCK は LOCK でロックした範囲を解除します。

start を省略すると 1 とみなされます。TO end を省略すると、start 番号のレコードだけロックされます。

start と end を共に省略すると、ファイル全体をロックします。

シーケンシャル・ファイルでは start と end の指定は意味がなく、いつもファイル全体がロックされます。

LOCK, UNLOCK を機能させるためには、MS-DOS 上で一度 SHARE.EXE を実行しておかなければなりません。

例

LOCK #1, 1 TO 4

LOCK #1, 5 TO 8

...

UNLOCK #1, 1 TO 4

UNLOCK #1, 5 TO 8

.....まとめて、UNLOCK #1, 1 TO 8としてはいけない。LOCKで指定したのと同じレコード番号を使わなければならない。

LOF

(ファイル・サイズの取得)

関数

書式

$$\text{LOF}(n)$$

↑ ファイル番号

機能

ファイル番号 `n` のファイルのサイズをバイト数で返します。

OPEN COM 文でオープンしたデバイスに LOF を使用すると、出力バッファ内の空きバイト数を返します。

TYPE Man

Namae AS STRING * 16

Year AS INTEGER

END TYPE

DIM A AS Man

と定義された Man 型のレコードをセーブした、ファイル番号 1 のファイルがあったとき、

MaxRec = LOF (1)/LEN(A)

はファイルのレコード総数を示します.

’ ---/* ファイル・サイズの取得 */---

```
TYPE Man
  Namae AS STRING * 16
  Year AS INTEGER
END TYPE
DIM a AS Man

OPEN "b:name.dat" FOR INPUT AS #1

PRINT "ファイルのバイト数 : "; LOF(1)
PRINT "ファイルのレコード数: "; LOF(1) / LEN(a)

CLOSE #1
```

ファイルのバイト数	: 72
ファイルのレコード数	: 4

LOG (自然対数)		関数
書 式	LOG(x)	
機 能	自然対数 log _e x を求めます.	

LPOS (印刷行バッファ内のプリンタ・ヘッドの現在位置の取得)		関数
書 式	LPOS(n) ↑ プリンタ番号	
機 能	n 番で示されるプリンタ (LPTn:) のヘッドの現在位置を取得します. LPOS 関数はタブ文字を展開しませんので、返す値は必ずしもプリンタヘッドの物理的位置とはなりません.	

例	<pre>’ ---/* 印字位置の取得 */--- FOR k = 48 TO 72 LPRINT CHR\$(k) IF LPOS(1) > 10 THEN LPRINT ←印字位置が10桁を越えたら改行 NEXT k LPRINT</pre> <div>0123456789 :;<=>?@ABC DEFGH</div>
---	---

LPRINT (プリンタへのデータ出力)

書 式 LPRINT arg1 {;|, }...
 ↑ 出力データ

機 能 arg1以下のデータをプリンタに出力します。データの区切りに“;”と“,”を用いた場合の違いは PRINT と同じです。
LPRINT はプリンタデバイス LPT1:を使用します。

LPRINT USING(書式制御付きのプリンタへのデータ出力)

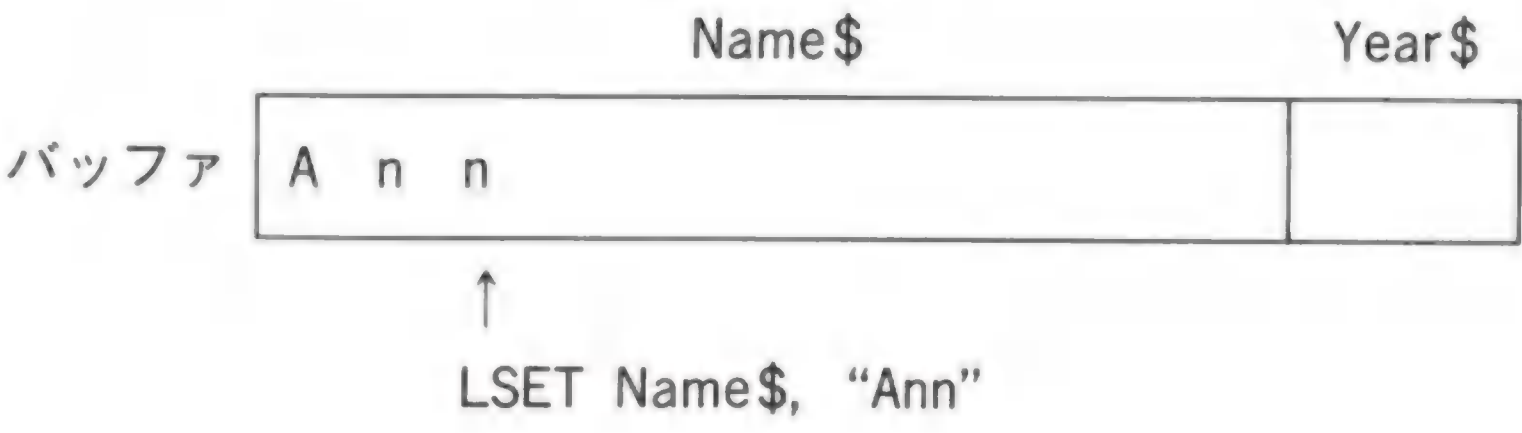
書 式 LPRINT USING format ; arg1{;|, }...
 ↑ 出力データ
 ↑ 書式制御文字列

機 能 arg1以下のデータを format で示される書式にしたがってプリンタに出力します。
format の指定方法は PRINT USING と同じです。

LSET (ファイル・バッファへの値の設定)

書 式 ① LSET strvar = strexpression
 ↑ 文字列
 ↑ FIELD で設定した文字列変数
 ② LSET recvar1 = recvar2
 ↑ レコード変数

機 能 ① FIELD 文で設定したリード/ライト・バッファに割り当てられている変数に文字列データを左づめにして格納します。余った部分は空白で埋められます。文字列が長ければ、はみ出した部分は捨てられます。
FIELD #1, 16 AS Name\$, 2 AS Year\$



strvar には FIELD 文で設定した文字列変数ではなく、一般の固定長文字列変数を使用することができます。機能はフィールド・バッファへの設定と同じです。

②異なる型のレコードに対する代入を行います。

```
TYPE S2
  Name$ AS STRING * 2
END TYPE

TYPE S3
  Name$ AS STRING * 3
END TYPE
DIM A AS S2, B AS S3
:
LSET S3 = S2
```

同じ型のレコードに対しては単なる代入文で行えます。

例 MKD\$ 参照.

LTRIM\$

(左側スペースの削除)

関数

書 式 LTRIM\$(str)
 ↑文字列

機 能 文字列 str の左側にあるスペースを削除します。スペースは、1 バイト文字、2 バイト文字のスペースです。

例 A\$ =" Hello BASIC"
 B\$ = LTRIM\$(A\$) ... "Hello BASIC"を返す。

MID\$

(中間文字列の取得)

関数

書 式 MID\$(str, start[, n])
 ↑ ↑ ↑
 文字列 取り出す開始位置 取り出す文字数

機 能 文字列 str の start 位置から n 個の文字を取り出して返します。n を省略すると文字列の終わりまで取り出します。
日本語を含む文字列には KMID\$ を使用してください。

例 A\$ = "Hello BASIC"
 B\$ = MID\$(A\$, 7, 3) ... "BAS"が返される。
 C\$ = MID\$(A\$, 7) ... "BASIC"が返される。

MID\$

(文字列の置き換え)

書 式

MID\$(strvar, start[, n])= strexpression

↑ ↑ ↑ ↑

置き換えられる文字列 置き換え開始位置 置き換える文字数 置き換える文字列

機 能

文字列変数 strvar の start 位置以後の文字を strexpression で置き換えます。n を指定すると、strexpression の先頭から n 文字分を置き換えます。置き換える文字列が長すぎる場合は、はみ出た文字は捨てられます。日本語を含む文字列には KMID\$ を使用してください。

例

```
A$ = "This is a pen"
MID$(A$, 11) = "book"
```

... A\$は"This is a boo"となる。k は捨てられることに注意。

MKD\$/MKI\$/MKL\$/MKS\$

(数値を内部コード形式の文字列に変換)

関数

書 式

MKD\$(double) ← 倍精度実数値を 8 バイト文字列に変換
 MKI\$(integer) ← 整数値を 2 バイト文字列に変換
 MKL\$(long) ← 倍長整数値を 4 バイト文字列に変換
 MKS\$(single) ← 単精度実数値を 4 バイト文字列に変換

機 能

FIELD で定義したリード/ライト・バッファは文字列型なので、ここに数値データを格納する場合は、MKD\$/MKI\$/MKL\$/MKS\$関数を用いて文字列に変換してから行います。

例

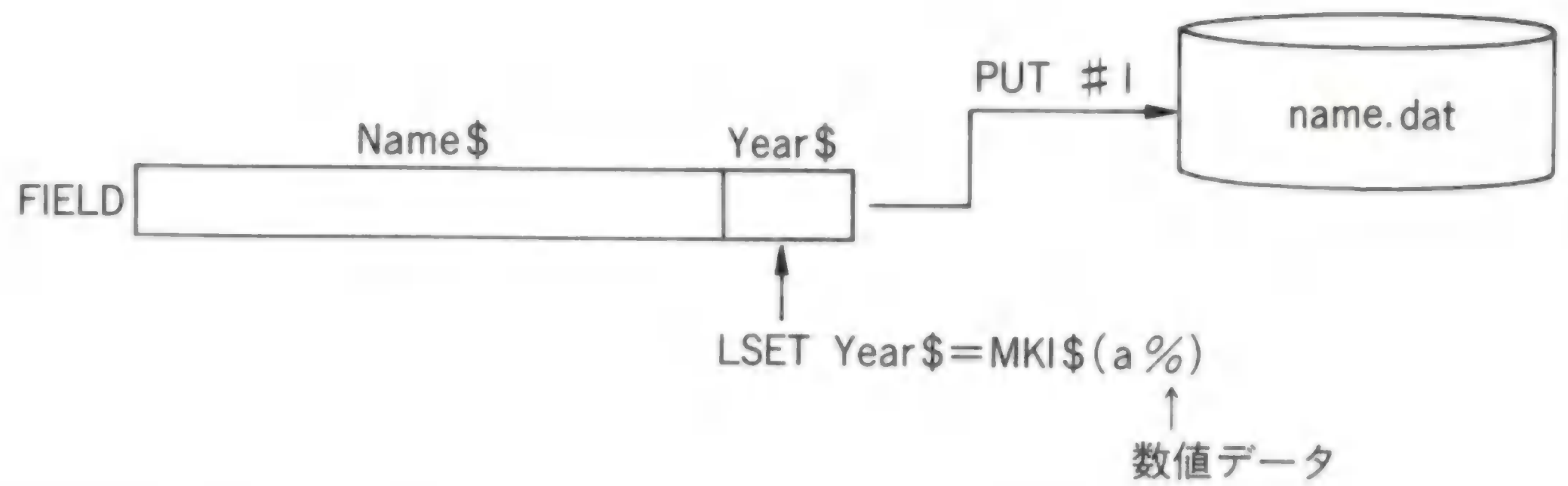
```

' ---/* ランダム・ファイルの書き込み */---
OPEN "b:name.dat" FOR RANDOM AS #1 LEN = 18
FIELD #1, 16 AS Name$, 2 AS Year$

INPUT "Rec No "; Rec
WHILE Rec > 0
    INPUT "name . year "; n$, a%
    LSET Name$ = n$
    LSET Year$ = MKI$(a%)
    PUT #1, Rec
    INPUT "Rec No "; Rec
WEND
CLOSE #1
```

```

Rec No ? 3
name . year ? Ann,19
Rec No ? 1
name . year ? Candy,21
Rec No ? 4
name . year ? Rolla,20
Rec No ? 0
```

MKDIR

(サブディレクトリの作成)

書 式

MKDIR path
 ↳ サブディレクトリ名

機 能

path で示されるサブディレクトリを作成します。

例

MKDIR "B:¥SATO¥WORK"

… ドライブ B のサブディレクトリ SATO の下にサブディレクトリ WORK を作る。サブディレクトリ SATO はすでに作成されていない。

MKSMBF\$/MKDMBF\$

(IEEE 形式の数値を MBF 文字列に変換)

関数

書 式

MKSMBF\$(single) ← 単精度実数値を 4 バイト文字列に変換
 MKDMBF\$(double) ← 倍精度実数値を 8 バイト文字列に変換

機 能

IEEE 形式の数値を マイクロソフト・バイナリ 形式 (MBF) の文字列に変換します。

NAME

(ファイル名の変更)

書 式

NAME old AS new
 ↳ 新しいファイル名
 ↳ 古いファイル名

機 能

old で示されるファイル名を new に変更します。
 old と new のパスが異なる場合は、サブディレクトリ間の移動となります。異なるドライブ間の移動はできません。

例

```
NAME "abc.bas" AS "aaa.bas"
... abc.bas を aaa.bas にリネームする。
NAME "abc.bas" AS "work¥aaa.bas"
... カレントディレクトリの abc.bas を、サブディレクトリ work に
aaa.bas として移動する。カレントディレクトリの abc.bas はな
くなる。
```

OCT\$ (数値を 8 進文字列に変換) 関数

書式

```
OCT$(n)
      ↑ 数値
```

機能

数値 n を 8 進文字列にして返します。

例

```
OCT$(255) ... 377を返す。
```

ON ERROR GOTO (エラー・トラッピングルーチンの設定)

書式

```
ON ERROR GOTO line
      ↑ トラッピングルーチンの入口
```

機能

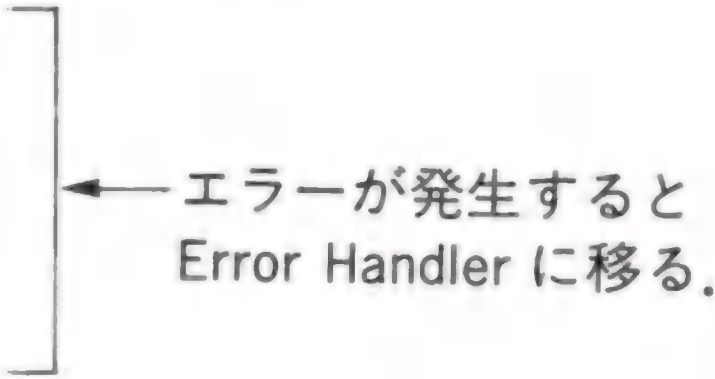
エラー・トラッピングルーチンのエントリポイントを設定します。
line はエントリポイントを示すラベルまたは行番号です。
line に 0 を指定すると、エラー・トラッピング処理を停止します。つまりエラーが発生すると、エラーメッセージが表示されプログラムの実行は停止します。

例

```
ON ERROR GOTO ErrorHandler ←———トラッピングルーチンの設定

FOR k = 1 TO 5
  READ x
  PRINT x, LOG(x)
NEXT k
DATA 2, -3, 5, -1, 6
END

ErrorHandler:
  PRINT x; "Arg is error"
RESUME NEXT ←———エラーのあった文の次の文に移す。
```



2	.6931472
-3	-3 Arg is error
5	1.609438
-1	-1 Arg is error
6	1.791759

ON event GOSUB (イベント・トラッピングルーチンの設定)

書 式 ON event GOSUB line
 ↑ イベント ↑ トラッピングルーチンの入口

機 能 event で示されるイベントが発生したときに分岐するサブルーチンを設定します。line はラベルまたは行番号です。
event として次の3つが指定できます。

event	機 能
COM(n)	n で示される通信ポートに文字を受信したときにイベント・トラッピング。n は1または2
KEY(n)	n 番のキーが押されたときにイベント・トラッピング。n は1～25で、対応するキーはファンクション・キー, 方向キー, ユーザ定義キー。詳しくは KEY(n) の項を参照
TIMER(n)	n 秒経過したときにイベント・トラッピング n は1～86400秒(24時間)

ON event GOSUB 文は、イベント・トラッピングルーチンを設定するだけです。イベント・トラッピングを許可/禁止/停止するには、次の文で行います。

- event ON イベント・トラッピングを許可します。
- event OFF イベント・トラッピングを行いません。
- event STOP イベント・トラッピングを停止します。停止中に発生したイベントは記憶されています。

ON event 文を含むプログラムをコンパイラ(bc)でコンパイルする際には、/v または/w オプションを指定してください。

例

```
' ---/* 時計の表示 */---
ON TIMER(1) GOSUB Clock
TIMER ON ← イベントトラッピングの許可

CLS
WHILE 1
  PRINT ".": ← 画面に.を表示し続ける
  oWEND
END

Clock:      ' 時間表示ルーチン
  x = POS(0): y = CSRLIN ' カーソル位置の取得
  LOCATE 1, 70
  PRINT TIMES
  LOCATE y, x          ' カーソル位置の復帰
  RETURN
                                     ...画面の右上隅に1秒おきに時刻を表示する
```


ON ~ GOSUB (条件によるサブルーチンコール)

書 式

ON n GOSUB line1, line2, ...

↑ 式 ↑ サブルーチンの入口

機能

式 n の値が 1 なら line1 で示されるサブルーチン, 2 なら line2 で示されるサブルーチン, ... へ分岐します.

line は行番号またはラベルです.

例

ON n GOSUB ABC,AAA,XYZ

```

graph TD
    ABC[ABC:  
RETURN]
    AAA[AAA:  
RETURN]
    XYZ[XYZ:  
RETURN]
    ABC --- AAA
    AAA --- XYZ
    
```

…nが1ならABC, 2ならAAA, 3ならXYZのサブルーチンへ分岐する.

ON ~ GOTO (条件による分岐)

書 式

ON n GOTO line1, line2, ...

↑ 式

↑ 飛び先

機能

式 n の値が 1 なら line1, 2 なら line2, ...へ分岐します.
line は行番号またはラベルです.

例

ON n GOTO ABC, AAA, XYZ

・・・n が 1 なら ABC, 2 なら AAA, 3 なら XYZ へ分岐する.

(ファイルのオープン)

書式

共有モード →

① OPEN filename [FOR mode][ACCESS access][lock] AS [#]n

↑ ファイル名 ↑ ファイルモード ↑ [LEN = reclen] ↑ レコード・サイズ

↑ アクセスモード

↑ ファイル番号

② OPEN mode, [#]n, filename [, reclen]

↑ ファイルモード ↑ ファイル番号 ↑ ファイル名 ↑ レコード・サイズ

機能

① filename で示されるファイルをファイル番号 n でオープンします。
mode はファイルのオープンモードで以下のものを指定します。

mode	機能
OUTPUT	シーケンシャルファイルの出力モード
INPUT	シーケンシャルファイルの入力モード
APPEND	シーケンシャルファイルのアペンドモード
RANDOM	ランダムアクセスモード
BINARY	バイナリモード

access はファイルのリード/ライトアクセスのモードを設定するもので以下のものを指定します。

access	機 能
READ	読み出し専用モード
WRITE	書き込み専用モード
READ WRITE	読み出し, 書き込み両用モード

lock はマルチプロセシング環境で意味を持ち、オープンファイルへの他のプロセスからのアクセスを制限するものです。

lock	機 能
指定せず	他のプロセスからアクセスすることはできない
SHARED	他のプロセスから読み出しモードでも書き込みモードでもアクセスできる
LOCK READ	他のプロセスからこのファイルを読み出すことを禁止する
LOCK WRITE	他のプロセスからこのファイルに書き込むことを禁止する
LOCK READ WRITE	他のプロセスからこのファイルに読み出すこと,書き込むことを禁止する

reclen はランダム・ファイルのレコード長(バイト数)を指定します。シーケンシャルファイルで reclen を指定した場合は、リード/ライト用バッファのサイズとなります。デフォルトのバッファサイズは 512 バイトです。

- ② filename で示されるファイルをファイル番号 n でオープンします。この書式は他の BASIC との互換性を保つためのものです。
mode はファイルのオープンモードで以下のものを指定します。

mode	機 能
O	シーケンシャルファイルの出力モード
I	シーケンシャルファイルの入力モード
A	シーケンシャルファイルのアペンドモード
R	ランダムアクセスモード
B	バイナリモード

reclen の意味は①の書式の場合と同じです。
①, ②の書式とも、filename に次のようなデバイスファイルを指定できます。

- KYBD: キーボード
SCRN: スクリーン
COMn: 通信ポート (RS-232C)
LPTn: プリンタ
CONS: 標準出力

例

```
OPEN "abc.dat" FOR APPEND AS #1
```

… ファイル abc.dat をアペンドモードでオープンする、つまりすでにあるデータの終わりからデータを書き出すモード。



TYPE Man

Namae AS STRING * 16

Year AS INTEGER

END TYPE

DIM A AS Man

```
OPEN "abc.rnd" FOR RANDOM AS #1 LEN = LEN(A)
```

・・・Man 型のデータをレコードとするランダム・ファイルのオープン

```
OPEN "LPT1:BIN" FOR OUTPUT AS #1
```

…プリンタデバイスをバイナリモードの出力モードでオープン

```
OPEN  "abc.rnd" FOR RANDOM ACCESS READ LOCK WRITE
                                           AS #1
```

- abc.rnd をランダム・ファイルの読み出し専用ファイルとしてオープン。このファイルはほかのプロセスからの書き込みを禁止している。

OPEN COM

(通信ポートのオープン)

書式

OPEN "COMn: optlist1, optlist2" [FOR mode] AS[#]n [LEN = reclen]

↑ ↑ ↑ ↑

オプションリスト ファイル レコード・
サイズ

ポート番号(1, 2) モード 番号

機能

通信ポートの COM1または COM2をオープンします。
optlist1は次の形式です。

```
[speed][,[parity][,[data][,[stop]]]]
```

オプション	機能
speed	ボーレート. 9600, 4800, 2400, 1200, 600, 300, 150, 75
parity	パリティチェック. N…チェックしない, E…偶数パリティ, O…奇数パリティ
data	データビット長. 5, 6, 7, 8
stop	ストップビット長. 1, 2

optlist2に指定できるオプションは以下のとおりで、複数指定するときはカンマ(,) で区切ってください。

オプション	機能
ASC	アスキーモードでオープン <ul style="list-style-type: none">・タブをスペースに展開・行末にキャリッジリターンコードを付加・<code>CTRL</code> + <code>Z</code> をファイルの終わりとして処理する・XON/XOFF 制御を許可する・通信チャネルをクローズしたときに <code>CTRL</code> + <code>Z</code> コードを送信
BIN	バイナリモードでオープン <ul style="list-style-type: none">・タブを展開しない・行末にキャリッジリターンコードを付加しない・<code>CTRL</code> + <code>Z</code> をファイルの終わりとして処理しない・通信チャネルをクローズしたときに <code>CTRL</code> + <code>Z</code> コードを送信しない
CD[m]	m ミリセカンド過ぎても DCD (Data Carrier Detect) が上位にならないときデバイスのタイムアウトが起こる PC-9801シリーズでは m=0とする
CS[m]	m ミリセカンド過ぎても CTS (Clear To Send) が上位にならないとき、デバイスのタイムアウトが起こる PC-9801シリーズでは m=0とする
DS[m]	m ミリセカンド過ぎても DSR (Data Set Ready) が上位にならないときデバイスのタイムアウトが起こる
LF	キャリッジリターン文字(ODH)のあとにラインフィード文字(OAH)を自動的に付け加える
OP[m]	オープンに成功するまでの待ち時間を m ミリセカンドに設定する.m を省略すると待ち時間は無期限となる.OP を指定しないと CD と DS で設定した m の長い方に10倍した時間待つ
RB[n]	受信バッファのサイズを n バイトに設定 デフォルトのサイズは512バイト
RS	RTS (Request To Send) の検知を抑制する

mode は次のものを指定します。

mode	機能
OUTPUT INPUT RANDOM	シーケンシャル出力モード シーケンシャル入力モード ランダムアクセスモード

例

次のプログラムは2台のコンピュータ間でRS-232Cを介してデータの送受信を行うものです.

・送信側プログラム

OPEN "COM1:9600,N,8,1,ASC" FOR OUTPUT AS #1

DO

INPUT a\$

PRINT #1. a\$

```
LOOP WHILE a$ <> "end" ← endという文字列を受信すると終わり
CLOSE #1
```

CLOSE #1

・受信側プログラム

```
OPEN "COM1:9600,N,8,1,ASC" FOR INPUT AS #1
```

INPUT #1. a\$

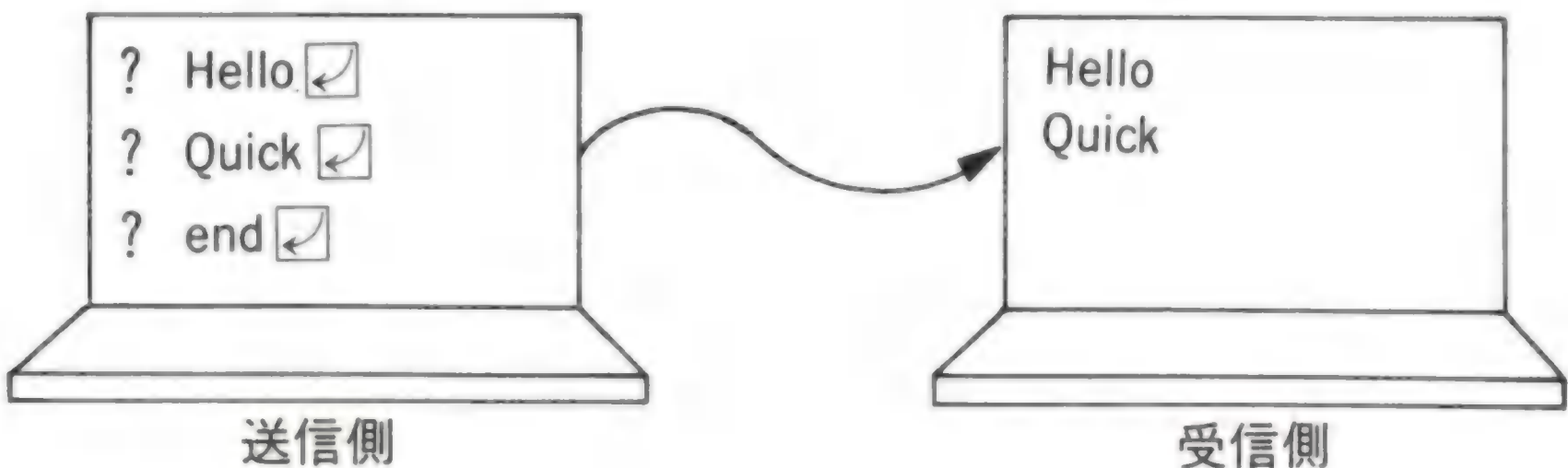
DO WHILE a\$ <> "end"

PRINT a\$

INPUT #1. a\$

LOOP

CLOSE #1



OPTION BASE (配列添字の下限デフォルト値の設定)

(配列添字の下限デフォルト値の設定)

書式

OPTION BASE n

↑ 添字の下限

機能

配列添字の下限デフォルト値を n (0 または 1) に指定します。
OPTION BASE を行わなければ、配列添字の下限デフォルト値は 0 となります。

OPTION BASE 文は、モジュール内の配列を宣言する前に一度しか使用できません。

例

OPTION BASE 1

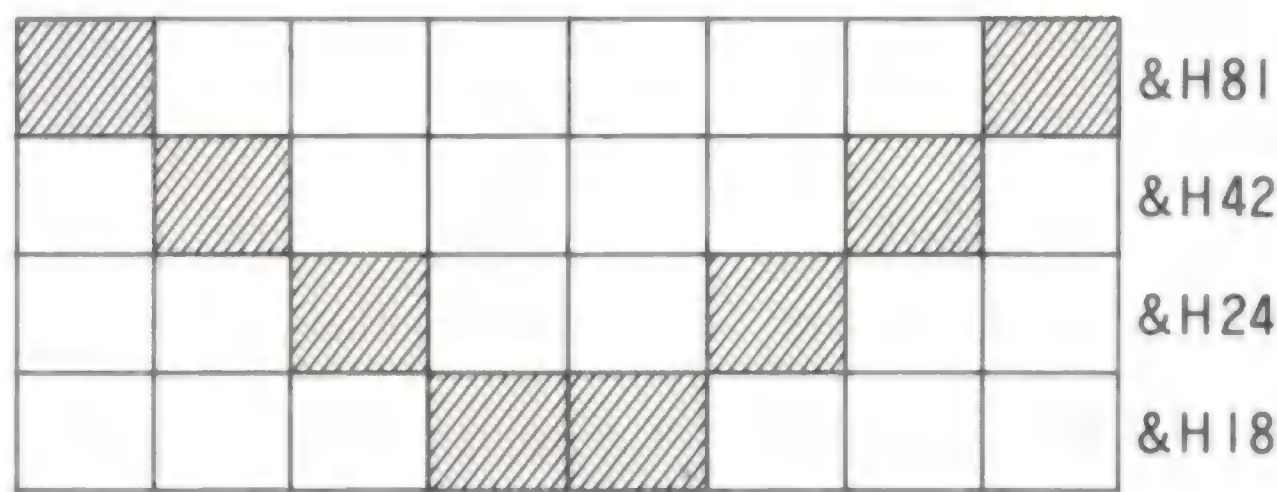
DIM A(20)

…A(1)～A(20)の配列が確保される。

つまり、タイルパターンを構成する文字列データ Tile\$ は、次のようになります。

Tile\$ = CHR\$(Blue) + CHR\$(Green) + CHR\$(Red) + CHR\$(intensity)

それでは次のようなパターンを紫色でタイリングするための文字列 Tile\$ を示します。



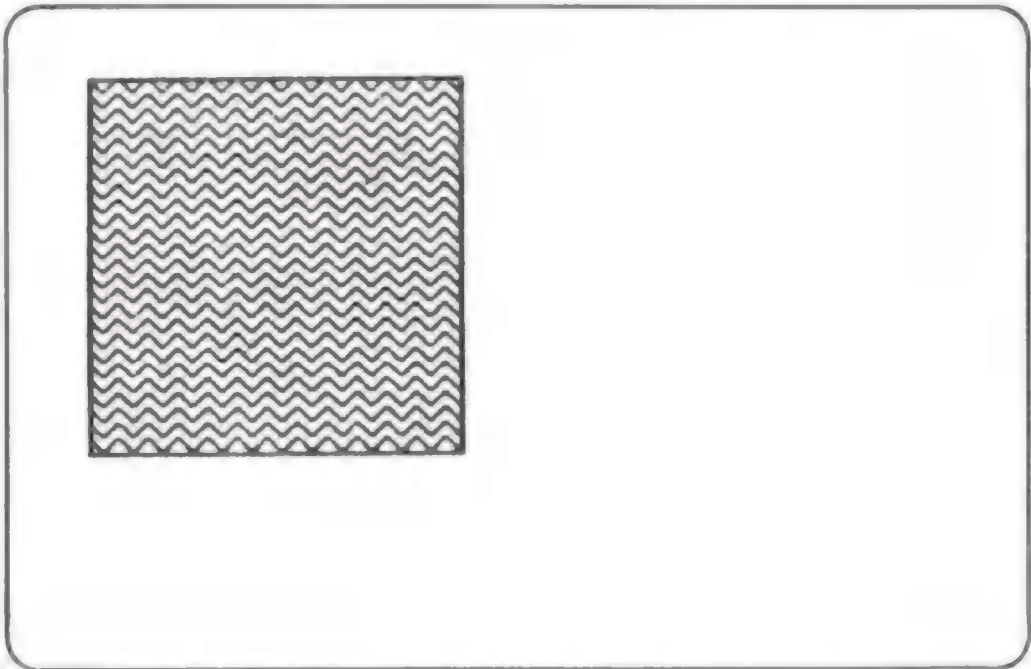
Tile\$ = CHR\$(&H81) + CHR\$(0) + CHR\$(&H81) + CHR\$(&HFF) ← 1 段目
+ CHR\$(&H42) + CHR\$(0) + CHR\$(&H42) + CHR\$(&HFF) ← 2 段目
+ CHR\$(&H24) + CHR\$(0) + CHR\$(&H24) + CHR\$(&HFF) ← 3 段目
+ CHR\$(&H18) + CHR\$(0) + CHR\$(&H18) + CHR\$(&HFF) ← 4 段目

↑ ↑ ↑ ↑
Blue のデータ Green のデータ Red のデータ 輝度のデータ

例

```
' ---/* タイリング */---  
  
SCREEN 0: CLS  
  
Tile$ = ""  
FOR k = 1 TO 16  
  READ a$  
  a = VAL("&h" + a$)  
  Tile$ = Tile$ + CHR$(a)  
NEXT k  
DATA 81,81,00,ff  
DATA 42,42,00,ff  
DATA 24,24,00,ff  
DATA 18,18,00,ff  
LINE (0, 0)-(100, 100), 7, B  
PAINT (50, 50), Tile$, 7
```

Red のデータ
輝度のデータ
Green のデータ
Blue のデータ



PALETTE/PALETTE USING
(パレットの設定)

書 式

- ① PALETTE [attribute, color]
 ↑ ↑
 属性番号 色番号
- ② PALETTE USING arrayname[(n)]
 ↑ ↑
 配列名 配列の要素番号

機 能

- ① attribute で示される属性番号に、color で示される色を割り当てます。パラメータを省略すると、attribute と color の関係はデフォルト値に設定されます。
- ② 属性番号 1 ～16 に対し、配列 arrayname(n), arrayname(n+1) … arrayname(n+15) に格納されている色番号をそれぞれ設定します。添字の n を省略すると、配列の先頭要素から採用されていきます。配列のサイズは、n 番目の要素以後に属性番号の数(最大で16)以上なければなりません。

PALETTE 文により、属性番号がほかの色番号に変更されると、すでにその属性番号で描画されている図形の色は、新たに設定された色番号の色に一瞬にして変わります。

使用できる色番号はデジタル機とアナログ機、さらにアナログ機においてはパレット・モードにより異なります。

●デジタル機の場合

色番号、属性番号ともに8種(0～7)です。

0	1	2	3	4	5	6	7
黒	青	緑	水	赤	紫	黄	白

●アナログ機の場合

SCREEN 文のパレット・モード 1 ～ 3 に対応して、16色、64色、4096色を使うことができます。属性番号はいずれの場合も16種(0～15)です。

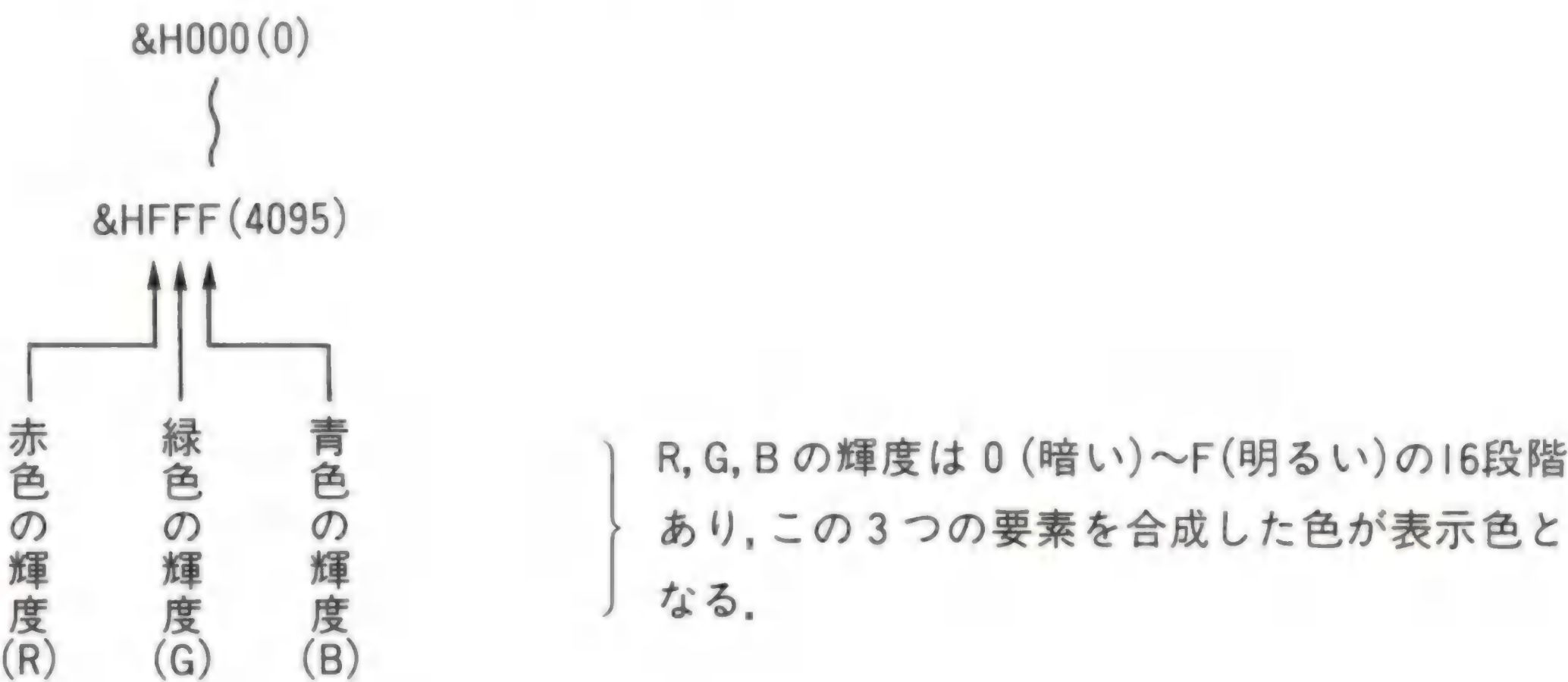
・16色(パレット・モード1)

0	1	2	3	4	5	6	7	8～15
黒	青	緑	水	赤	紫	黄	白	0～7の暗い色

・ 64色(パレット・モード 2)

0～ 7	黒, 青, 緑, 水, 赤, 紫, 黄, 白
8～15	(0～7) + 薄い青
16～23	(0～7) + 薄い緑
24～31	(0～7) + 薄い水色
32～39	(0～7) + 薄い赤
40～47	(0～7) + 薄い紫
48～55	(0～7) + 薄い黄色
56～63	(0～7) + 薄い白

・ 4096色(パレット・モード 3)

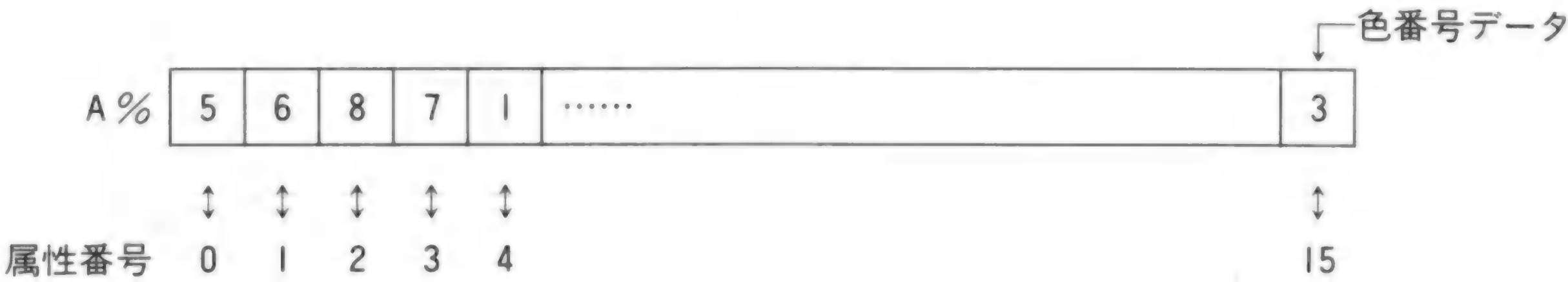


例

PALETTE 1, 2 ... 属性番号 1 に色番号 2 (緑)を割り当てる。すでに属性番号 1 で描画されている図形は一瞬に緑色に変わる。

PALETTE USING A%

... A%(0) のデータ 5 が属性番号 0 に, A%(1) のデータ 6 が属性番号 1 に, ... 以下同様に割り当てられる。




```
' ---/* 回るパレット */---

SCREEN 0: CLS

FOR k = 1 TO 7      ' 四角形を描く
    LINE (40 * k, 40)-(40 * k + 40, 80), k, BF
NEXT k

c = 0
DO
    a$ = INKEY$
    IF a$ <> "" THEN      ' キー入力があれば
        FOR i = 1 TO 7
            PALETTE i, ((c + i) MOD 7) + 1 ← 属性番号と色番号の対応を
                                                1つずらす.
        NEXT i
        c = c + 1
    END IF
LOOP WHILE a$ <> CHR$(&H1B)      ' E S C キーで終わり

PALETTE ← 属性番号と色番号の対応をデフォルト値に設定
```

… このプログラムでは、属性番号に対応する色番号を下図に示すように1つずつシフトさせます。これにより、7つの四角形の色が右から左に移動していくように見えます。

パレット(属性番号)							
	1	2	3	4	5	6	7
初期	1(青)	2()	3()	4()	5()	6()	7(白)
C=0	2	3	4	5	6	7	1
C=1	3	4	5	6	7	1	2
	⋮						

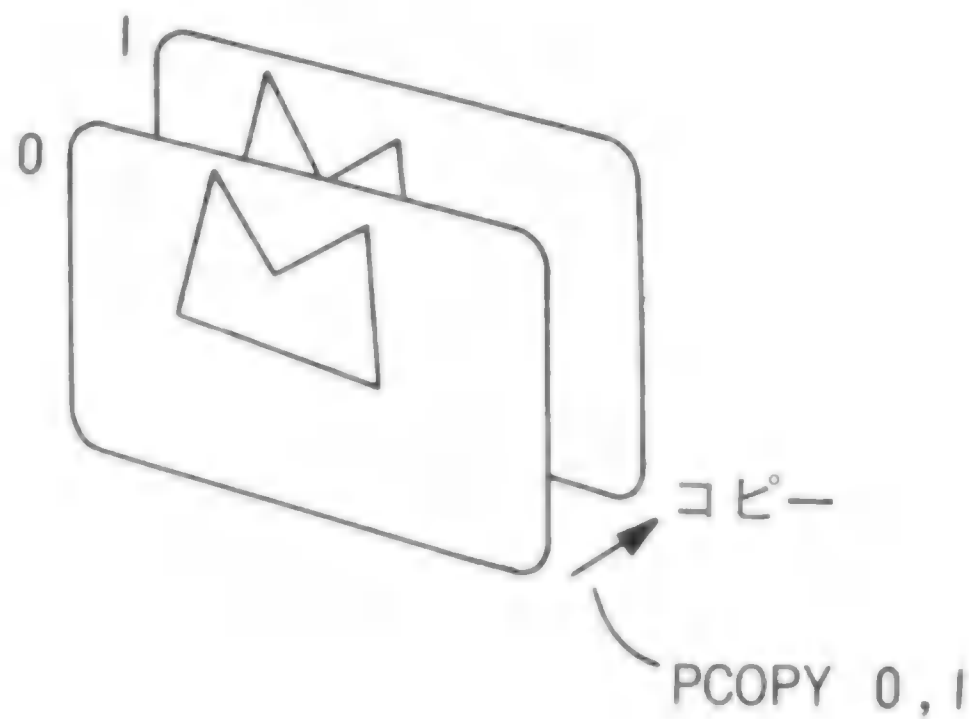
PCOPY (スクリーンページを別のページにコピー)

書 式

PCOPY source, dest
 ↑ ↑
 送り先のページ番号
 送り元のページ番号

機 能

source で示される画面ページのデータを，dest で示される画面ページにコピーします。もとの画面ページのデータはそのままです。



PEEK

(メモリから1バイトをリード)

関数

書式

PEEK(offset)

↑ オフセット値

機能

offset で示されるメモリ・アドレスの1バイトデータをリードします。

offset は DEF SEG 文で設定されている、セグメント・アドレスからのオフセット値です。

例

・ テキスト画面の直接アクセス

```
DEF SEG = &HA000
```

```
FOR k = 0 TO 80 * 25 * 2 - 1 STEP 2
```

```
  d = PEEK(k)
```

```
  IF ASC("A") <= d AND d <= ASC("Z") THEN
```

```
    d = d + (ASC("a") - ASC("A"))
```

```
    POKE k, d
```

```
  END IF
```

```
NEXT k
```

… PC-9800シリーズのテキスト VRAM は以下のようにになっている。

	1	2	桁	80
1	A0000	A0002		A009E
2	A00A0	A00A2		
↑	アスキー文字の表示は偶数番地だけを使用し、漢字の表示は偶数番地と奇数番地を使用する			
25	A0F80			A0FFE

このプログラムは、PEEK で VRAM 上のデータをリードし、その文字が大文字なら小文字に変換して、VRAM 上に書き直すものです。

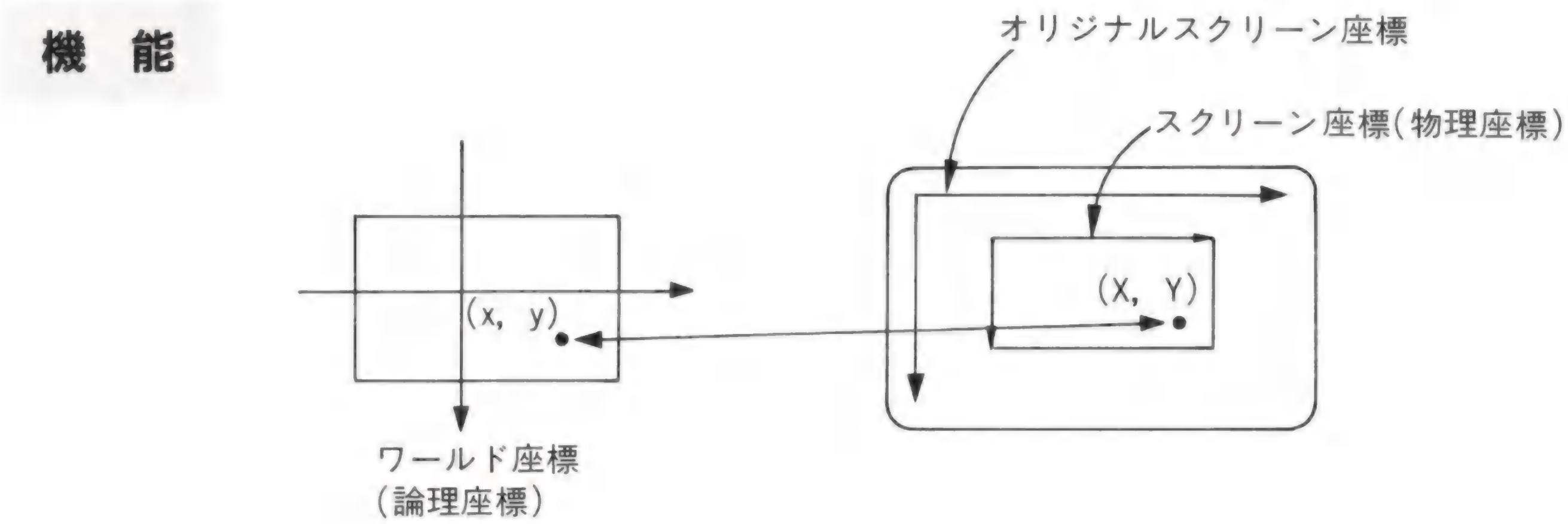
PMAP

(論理座標 \longleftrightarrow 物理座標の変換)

関数

書式

PMAP(position, mode)
 ↑ ↑
 座標 変換モード

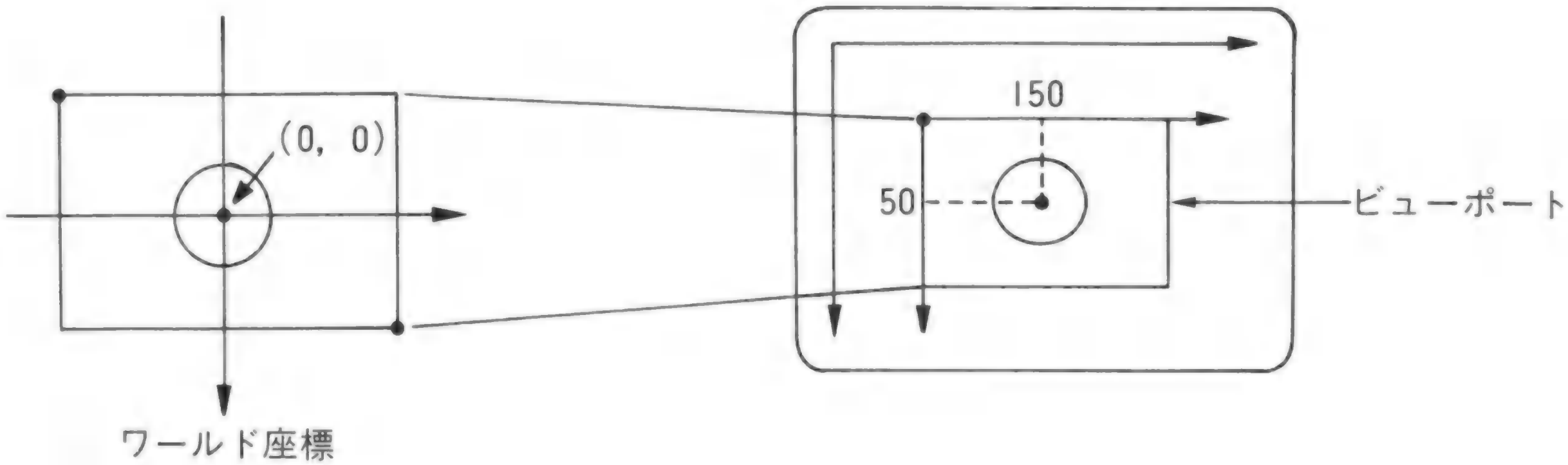


論理座標 (x, y) の物理座標上の位置 (X, Y) を求めたり、その逆に物理座標 (X, Y) の論理座標上の位置 (x, y) を求めます。
mode により次のような変換が行われます。

mode	機能
0	論理座標 x → 物理座標 X に変換
1	論理座標 y → 物理座標 Y に変換
2	物理座標 X → 論理座標 x に変換
3	物理座標 Y → 論理座標 y に変換

例

```
---/* ワールド座標をスクリーン座標に変換 */---  
  
SCREEN 0: CLS  
WINDOW (-320, -320)-(320, 320)  
VIEW (100, 100)-(400, 300)  
  
LINE (-320, -320)-(320, 320), 1, B  
CIRCLE (0, 0), 100, 1  
  
PRINT "スクリーン座標 x = "; PMAP(0, 0)  
PRINT "スクリーン座標 y = "; PMAP(0, 1)
```



POINT

(指定位置の色の取得)

関数

書式

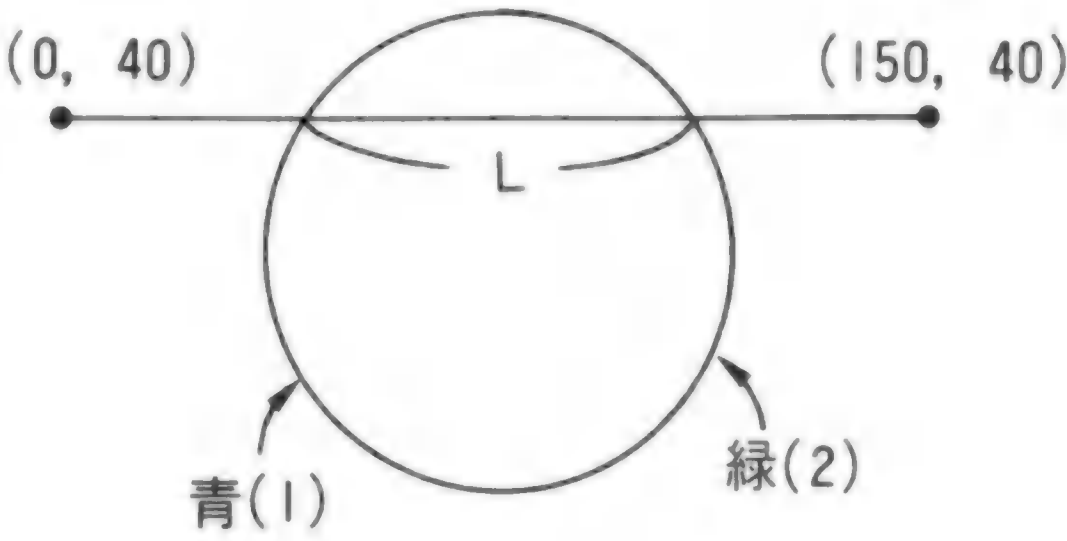
POINT (x, y)
↑ スクリーン座標

機能

スクリーン座標 (x, y) 位置のピクセルの属性番号を返します。

例

```
' ---/* 弦の長さを求める */---  
  
SCREEN 0: CLS  
rd = 3.14159 / 180  
CIRCLE (100, 50), 40, 1, 90 * rd, 270 * rd ← 青の半円  
CIRCLE (100, 50), 40, 2, 270 * rd, 90 * rd ← 緑の半円  
  
FOR x = 0 TO 150  
  c = POINT(x, 40)  
  IF c = 1 THEN xs = x  
  IF c = 2 THEN xe = x  
NEXT x  
PRINT "Length="; xe - xs ← 弦の長さ L
```



POINT

(描画現在位置の取得)

関数

書式

POINT (mode)
↑ 取得モード

機能

グラフィック画面への描画を行ったとき、描き終わった最後の座標は内部的に記憶されています。この座標を描画現在位置または LP (Last referenced point) 位置と呼びます。POINT は、この描画現在位置の座標を取得します。

mode	返される座標
0	物理座標での x
1	物理座標での y
2	論理座標での x
3	論理座標での y

例

```
' ---/* 描画現在位置の取得 */---  
  
SCREEN 0: CLS  
  
PRINT "最初の現在位置 "; POINT(0), POINT(1)  
  
LINE (100, 100)-(200, 200)  
PRINT "line後 "; POINT(0), POINT(1)  
  
CIRCLE (150, 150), 100  
PRINT "circle後 "; POINT(0), POINT(1)
```

最初の現在位置	320	200
line後	200	200
circle後	150	150

POKE (メモリに1バイトをライト)

書式 POKE offset, byte
↑ ↑
書き出すデータ
オフセット値

機能 offset で示されるメモリ・アドレスに、byte で示される 1 バイトのデータをライトします。
offset は DEF SEG 文で設定されているセグメント・アドレスからのオフセット値です。

例 PEEK 参照。

POS (現在のカーソル欄の取得) 関数

書式 POS(dummy)
↑
ダミーの数値引数。通常 0 を指定

機能 現在のカーソル位置の欄座標(x座標)を取得します。

例 CSRLIN 参照。

PRESET

(指定点のドットの消去)

書 式

PRESET [STEP](x, y)[,c]

↑ 表示色
↑ 指定点の座標

機 能

(x, y)位置に色属性 c の色のドットを付けます。c を省略すると、バックグラウンド色が採用されます。したがって、PRESET (x, y)は指定点のドットを消去することになります。

例

PSET 参照。

PRINT

(スクリーンへの出力)

書 式

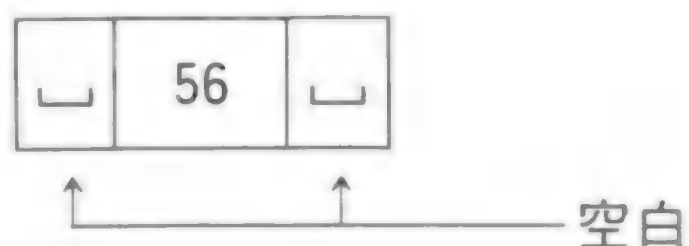
PRINT [arg{, |; }arg ...]

↑ 出力データ

機 能

arg で示されるデータをスクリーンへ出力します。

arg が数値データの場合、数字のうしろには必ず空白が付けられ、数字の前には、正の数値の場合は空白、負の数値の場合は負符号(−)が付けられます。



出力データが単精度実数および倍精度実数の場合、出力桁数に応じて固定小数点書式と浮動小数点書式の2通りの方法をとります。

● 固定小数点書式

7 桁以下の単精度実数、16桁以下の倍精度実数は次のような固定小数点書式で表示されます。

- 0000011
- 00000000000000011

● 浮動小数点書式

7 桁以上の単精度実数、16桁以上の倍精度実数は次のような浮動小数点書式で表示されます。

- 1.1E-07
- 1.1D-16

	書式制御文字	機 能
文字列の制御	!	文字列の最初の 1 文字だけを表示
	& _ _ ... &	&と&の間に置いた空白の数 n+ 2 個の文字を表示 文字列が n+ 2 より短いと文字列を左詰めにし、右側は空白となる
	@	可変長の文字列フィールドを意味し、引数の文字列はそのままの幅で表示される
数 値 の 制 御	##..., #...	表示桁数の指定。#の個数が整数部、小数部のそれぞれの桁数を示す #で指定した出力桁数より出力データの桁数が少なければ、数値は右詰めにして出力される
	+	数値の前後に、その数値の符号(+または-)を表示させる。+を書式制御文字の先頭に置けば数値の前に + を、書式制御文字列の末尾に置けば数値のあとに符号を表示する
	-	書式制御文字列の末尾に置き、負の数値の前に負符号を表示する
	* *	数値フィールドの空白を*で表示 * *は 2 桁分の桁数指定として数えられる
	¥¥	数値の直前に円記号を 1 つ表示。¥¥は 2 桁分の桁数指定として数えられる
	* *¥	数値フィールドの空白を*で表示し、数値の直前に円記号を表示 * *¥は 3 桁分の桁数指定として数えられる
	,	整数部データを 3 桁ごとにカンマ(,)で区切る。カンマは文字列中の小数点の左側に置く , は 1 桁分の桁数指定として数えられる
	^ ^ ^ ^ ^ ^ ^ ^ ^	E+xx のような指数形式を指定する E+xxx のような指数形式を指定する 数値の前後に+または-を指定しないと、小数点の左側の 1 桁が空白または負符号の表示にあてられる
	_(下線)	下線の右にある文字をそのまま表示する これは書式制御文字を表示するのに使う
	%	表示する数値が、指定した数値フィールドよりも長いと数値の前に%記号を表示

例

```
a = 100.56
b = 1234567
c$ = "Hello BASIC"
```

```
PRINT "....+....+....+....+....+...."
PRINT USING "###.##"; a
PRINT USING "#####"; a
PRINT USING "####.#"; a
PRINT USING "#.###^~^~"; a
PRINT USING "#.##"; a
```

```
PRINT
PRINT "....+....+....+....+....+...."
PRINT USING "+#####"; b
PRINT USING "#####"; b
PRINT USING "YY#####"; b
PRINT USING "#####."; b
```

```
PRINT
PRINT "....+....+....+....+....+...."
PRINT USING "& &"; c$
PRINT USING "!!"; c$
PRINT USING "@"; c$
```

```
PRINT
PRINT "....+....+....+....+....+...."
PRINT USING "#####.## #####.# #####."; a; a; a
PRINT USING "#####.##"; a; a; a
PRINT USING "合計=#####"; b
PRINT USING "& & #####"; c$; b
```

```
....+....+....+....+....+....
100.56
101
100.6
0.101E+3
%100.56

....+....+....+....+....+....
+1234567
***1234567
¥1234567
1,234,567

....+....+....+....+....+....
Hello
H
Hello BASIC

....+....+....+....+....+....
100.56 100.6 101.
100.56 100.56 100.56
合計= 1234567
Hello 1234567
```

PRINT #/ PRINT USING

(シーケンシャル・ファイルへのライト)

書 式

① PRINT #n , arg1 { , | ; } arg2 ...

↑ ↑
出力データ
ファイル番号

② PRINT #n , USING format; arg1 { , | ; } arg2 ...

↑ ↑ ↑
ファイル番号 書式制御文字 出力データ

機 能

PRINT, PRINT USING でスクリーンに表示したのと同じイメージでファイル番号 n のファイルにライトします。

区切り文字としてカンマ(,)を使用すると、データとデータの間の空白(14桁区切り)もファイルに書き込まれることに注意してください。

シーケンシャル・ファイルとして書き出すには、区切り記号として“,”も書き出さなければなりません。WRITE # を用いればこのような煩雑さはありません。

例

```
' ---/* シーケンシャル・ファイルの書き込み */---  
OPEN "man.dat" FOR OUTPUT AS #1  
  
INPUT "Name, year"; Name$, Year  
  
DO WHILE Year >= 0  
    PRINT #1, Name$: ", "; Year  
    INPUT "Name, Year"; Name$, Year  
LOOP  
CLOSE #1
```

区切りとして", "も書き出す。

```
Name, year? Ann, 18  
Name, Year? Candy, 19  
Name, Year? Eluza, 21  
Name, Year? Rolla, 20  
Name, Year? /, -1  
  
A>dump man.dat  
  
Dump Version 3.00  
  
00000000  41 6E 6E 2C 20 31 38 20-0D 0A 43 61 6E 64 79 2C  Ann, 18 ..Candy,  
00000010  20 31 39 20 0D 0A 45 6C-75 7A 61 2C 20 32 31 20  19 ..Eluza, 21  
00000020  0D 0A 52 6F 6C 6C 61 2C-20 32 30 20 0D 0A      ..Rolla, 20 ..
```

PSET

(指定点にドットの表示)

書 式

PSET [STEP] (x, y)[,c]

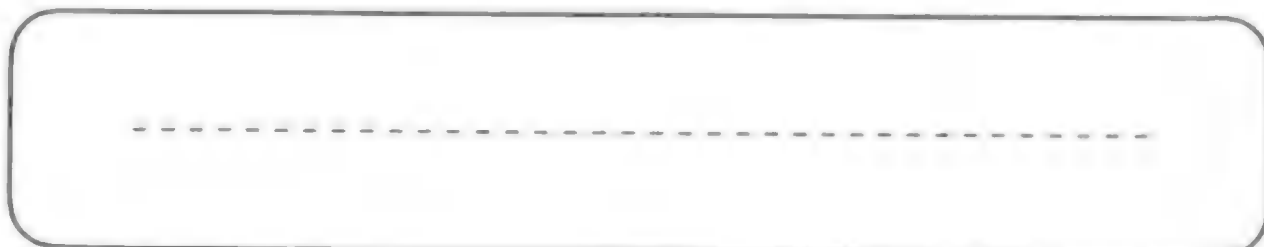
↑ 表示色
↑ 指定点の座標

機 能

(x, y)位置に属性番号cの色のドットを付けます。cを省略すると、COLOR 文で設定されている表示色が採用されます。

例

```
' ---/* PSET, PRESET */---  
  
SCREEN 0: CLS  
  
FOR x = 0 TO 200  
    PSET (x, 50), 2 ← 緑色でドットをセットしていく。結果として直線が引ける。  
NEXT x  
  
FOR x = 0 TO 200 STEP 5  
    PRESET (x, 50) ← セットされているドットを5つおきにリセットする。  
NEXT x  
結果として点線になる。
```



PUT

（保存イメージの描画）

書 式

```
PUT [STEP](x, y), var[(n)][, mode]
```

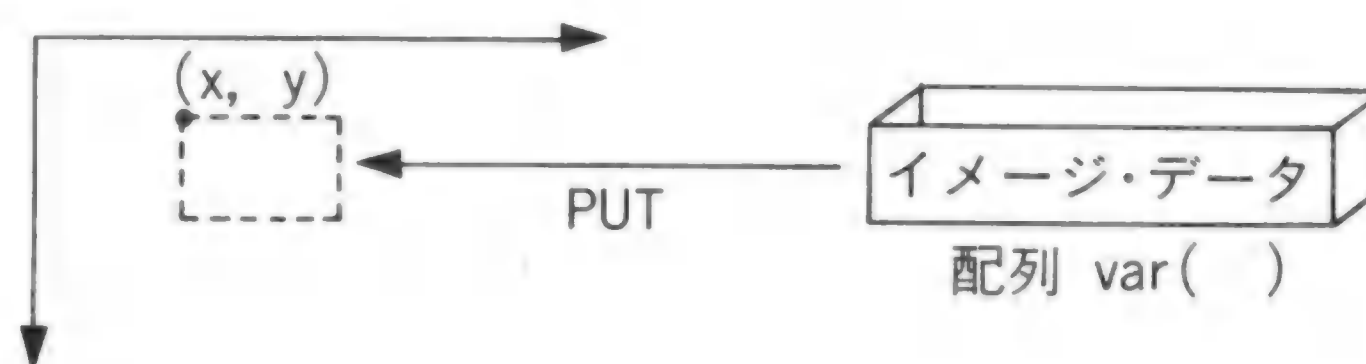
Diagram illustrating the relationship between the drawing mode, element number of the first data, arrangement name, and the coordinates of the top-left corner of the display area.

- 描画モード (Drawing Mode)
- 先頭データの要素番号 (Element Number of First Data)
- 配列名 (Arrangement Name)
- 表示域の左上隅の座標 (Coordinates of the Top-Left Corner of the Display Area)

機能

GET で配列 `var()` に取得してあるイメージ・データを、(x, y)を左上隅の座標とする領域に描画します。

n はイメージ・データが格納されている配列要素の先頭番号です。(n) を省略して配列名だけを書くと、配列の先頭要素からのデータが使われます。



mode は描画モードを指定するもので以下の 5 種類があります。
mode を省略すると XOR とみなされます。

mode	描 画 モ ー ド
PSET	配列に格納されているイメージ・データをそのまま画面に表示 その領域に前に描かれていたものはなくなる
PRESET	配列に格納されているイメージ・データを反転(たとえば白なら黒)して表示. その領域に前に描かれていたものはなくなる
AND	表示域にすでに描かれているデータとピクセルごとの AND をとって描く
OR	表示域にすでに描かれているデータとピクセルごとの OR をとって描く
XOR	表示域にすでに描かれているデータとピクセルごとの XOR をとって描く 背景色を消さずにイメージ・データを動かす場合に利用する

ピクセルごとの論理演算はプレーンごとに行われます。たとえば、反転(NOT)は次のように行われます。

NOT			
黒 (000)	↔	(111)	白
青 (001)	↔	(110)	黄
緑 (010)	↔	(101)	紫
赤 (100)	↔	(011)	水
<div>↑↑↑ Red Blue Green</div>			

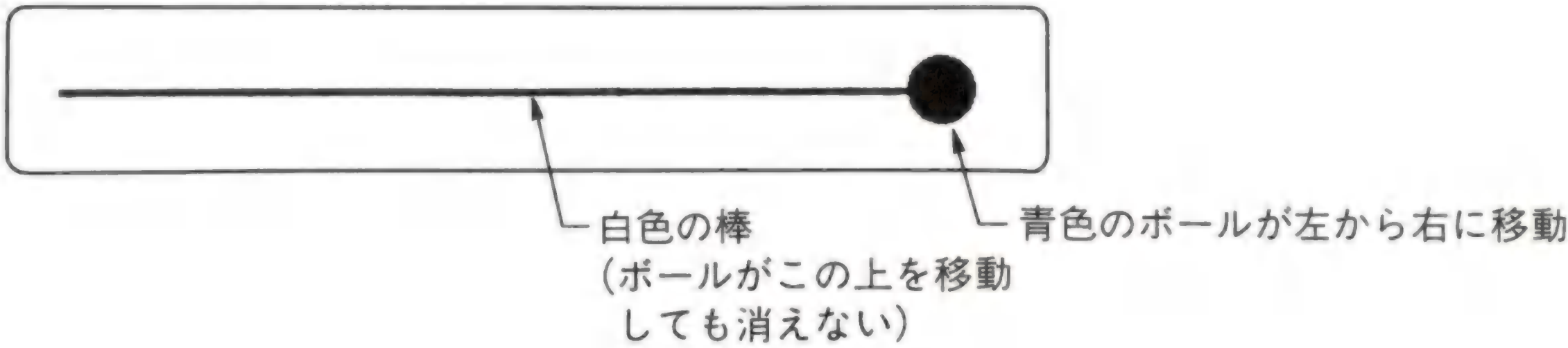
例

· ---/* 背景の中を移動させる */---

```
SCREEN 0: CLS
x = 45: y = 41
byte = 4 + 4 * ((x + 7) ¥ 8) * y
s = byte ¥ 2 + 1
DIM p%(s)

CIRCLE (20, 20), 15, 1: ' 円
PAINT (20, 20), 1, 1
GET (0, 0)-(44, 40), p%

CLS
LINE (0, 20)-(400, 22), 7, BF
PUT (0, 0), p%, XOR ← 最初の位置にボールを表示
FOR x = 0 TO 360 STEP 20
  PUT (x, 0), p%, XOR ← ボールを消す
  PUT (x + 20, 0), p%, XOR ← 次の位置にボールを表示
  FOR t = 1 TO 500: NEXT t
NEXT x
```



PUT

(ランダム・ファイルへのライト)

書 式

① PUT [#n][, rec]

$\begin{array}{c} \updownarrow \text{ファイル} \\ \text{番号} \end{array}$
 $\begin{array}{c} \updownarrow \text{レコード番号} \end{array}$

② PUT [#n],[rec][, var]

\uparrow ライト・データが格納されている変数

機 能

- ① FIELD 文で設定されているリード/ライト・バッファのデータをファイル番号 n のファイルにライトします。
- ② 変数 var のデータをファイル番号 n のファイルにライトします。var を指定した場合は、FIELD 文でリード/ライト・バッファを設定しないでください。

rec はランダム・ファイルでは読み出すレコード番号、バイナリ・ファイルでは読み出しを始めるバイト位置です。レコード番号、バイト位置の先頭番号は1です。

rec を省略すると、ファイル現在位置(最後に GET/PUT でアクセスしたレコードの次のデータ、または SEEK で指定したデータ位置)のデータをリードします。rec は $1 \sim 2^{31} - 1$ (2147483647) の範囲の値を指定します。

例

```
OPEN "b: name.dat" FOR RANDOM AS #1 LEN=18
```

```
FIELD #1, 16 AS Namae$, 2 AS Year$
```

```
  :
```

```
LSET Namae$, "Ann"
```

```
LSET Year$, MKI$(19)
```

```
PUT #1, Rec
```

← FIELD で設定したバッファのデータを
レコード番号 Rec としてライト

```
TYPE Man
```

```
  Namae AS STRING * 16
```

```
  Year AS INTEGER
```

```
END TYPE
```

```
DIM A AS Man
```

```
OPEN "b: name.dat" FOR RANDOM AS #1 LEN=LEN(A)
```

```
  :
```

```
A.Namae="Ann"
```

```
A.Year=19
```

```
PUT #1, Rec, A
```

←レコード型変数 A のデータをレコード番号 Rec
としてライト

RANDOMIZE

(乱数列の初期化)

書 式

RANDOMIZE [seed]
 ↑ 乱数列の種

機 能

seed で指定した値(-32768～32767)を使って、乱数ジェネレータを初期化します。

seed を省略すると、実行時に次のような問い合わせがあります。

新しい乱数列を入力してください(-32768 to 32767)->?

例

' ---/* サイコロ */---

RANDOMIZE TIMER ← 乱数列の初期化

FOR k = 1 TO 10

 m = INT(6 * RND + 1) ← 1 ～ 6 の整数乱数を発生

 PRINT m;

NEXT k

PRINT

3 3 1 1 6 6 6 1 4 5

READ

(プログラム中のデータを読む)

書 式

READ var, ...
 ↑ データを格納する変数

機 能

DATA 文で定義されているプログラム中のデータを変数 var にリードします。

変数とデータの型が一致していなければなりません。また READ 文で指定した変数の数に対しデータが少ない場合は、"Out of DATA"エラーが発生します。

例

READ A, B, C\$

READ D, E

⋮

DATA 5, 10, "Basic", 11, 6

…A に 5, B に 10, C\$に"Basic",
D に 11, E に 6 がリードされる。

REDIM (動的配列の割り当てスペースの変更)

書 式 REDIM [SHARED] var([lower TO] upper) [AS type], ...

↑ ↑ ↑ ↑
配列名 添字の下限値 添字の上限値 配列の型

機 能 動的配列のサイズを再度設定し直します。配列の次元を変更することはできません。

例 '\$DYNAMIC
DIM A(50, 50)
:
ERASE A ←A が割り当てられていた領域を解放し、
REDIM A(100, 100) ←再び100×100のサイズで割り当て直す。

REM (コメント)

書 式 ① REM comment
② ' comment

↑
コメント

機 能 コメントを記述します。他のステートメント(文)のうしろにコメントを書く場合、REM を用いると、ステートメントとの間にコロン(:)を必要としますが(')を用いると、コロン(:)は必要ありません。

例 Sum = 0:REM 初期化
Sum = 0' 初期化

RESET (すべてのディスク・ファイルのクローズ)

書 式 RESET

機 能 すべてのディスク・ファイルをクローズします。そのとき、バッファにあるデータはすべてファイルにライトされます。

RESTORE

(リードデータ位置の設定)

書 式

RESTORE [line]

↑ データのある行

機 能

READ 文が実行されると、DATA 文のデータのリード位置は次々に先に進んでいきます。RESTORE 文により、このリード位置を、指定の行位置のデータに設定することができます。

line は行番号またはラベルです。line を省略すると、プログラム中の最初の DATA 文の先頭データがリード位置となります。

例

RESTORE Boy

READ A\$, B\$

... A\$に Jimmy, B\$に Joe がリードされる。

⋮

RESTORE Girl

READ X\$, Y\$

... X\$に Ann, Y\$に Candy がリードされる。

⋮

Girl:

DATA Ann, Candy, Lisa, Eluza, Nancy

Boy:

DATA Jimmy, Joe, Bill, Steave, Tomy

・ ---/* カーレース */---

```
DATA " () "  
DATA "0-UU-0"  
DATA " UU "  
DATA "0-VV-0"  
DATA " "  
DATA " "  
DATA " "
```

WHILE 1

RESTORE

← READ 文で読むデータをプログラムの
最初の DATA 文のデータに設定する。

TB = INT(30 * RND + 2) ← 自動車の表示位置を乱数で求める。

FOR K = 1 TO 7

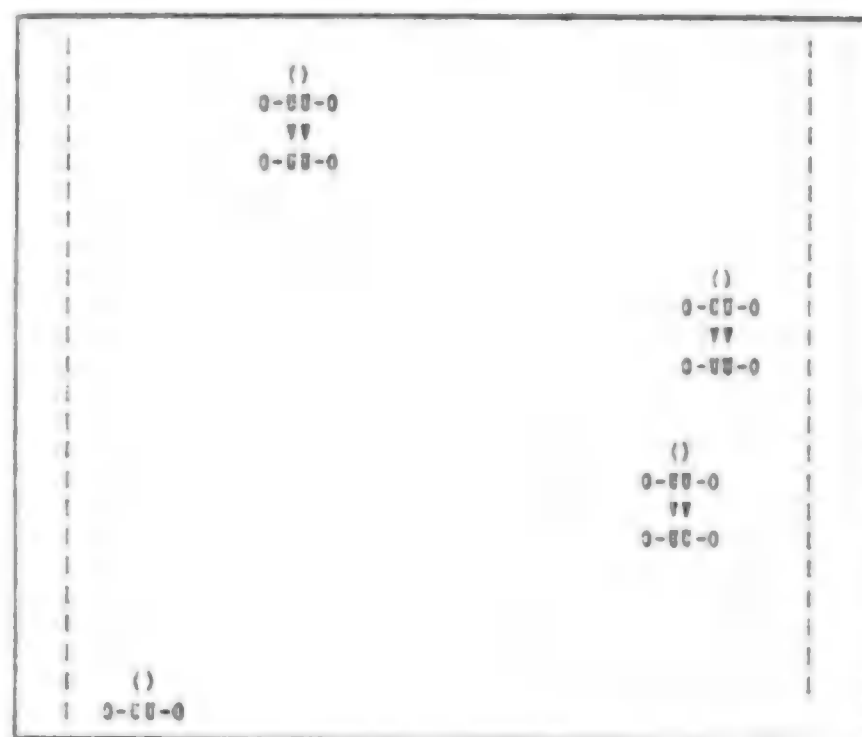
READ A\$

PRINT "I"; TAB(TB); A\$; TAB(38); "I"

NEXT K

WEND

END



RESUME(エラー・トラッピングルーチンからの復帰)**書 式**

- ① RESUME [0]
- ② RESUME NEXT
- ③ RESUME line
 ↑ 戻り先の行番号またはラベル

機 能

- ① エラーを生じた文に戻ります。
- ② エラーを生じた文の次の文に戻ります。
- ③ line で指定した行に戻ります。

例

ON ERROR GOTO 参照。

RETURN (サブルーチンからのリターン)**書 式**

RETURN [line]
 ↑ 戻り先の行番号またはラベル

機 能

GOSUB 文または ON event GOSUB 文の次の文に戻ります。line を指定すると、指定された行番号またはラベル位置に戻ります。
 RETURN はプロシージャからのリターンには使えません。プロシージャからのリターンには EXIT SUB, EXIT FUNCTION を使います。

例

GOSUB, ON event GOSUB

RIGHT\$ (右部分文字列の取得) 関数**書 式**

RIGHT\$(str, n)
 ↑ ↑
 ↑ 取り出す文字数
 ↑ 文字列

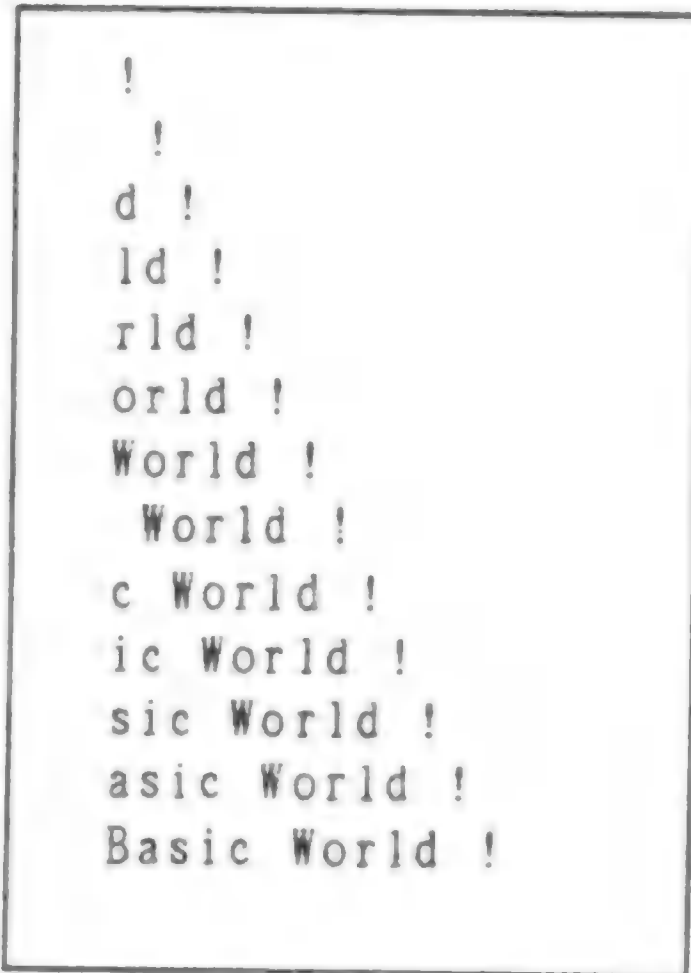
機 能

文字列 str の右端から n 文字(バイト)を取り出します。n が文字列の長さより大きければ文字列全体を返し、n が0ならヌル文字列を返します。

例

---/* 右部分文字列 */---

```
CLS
p$ = "Basic World !"
FOR k = 1 TO LEN(p$)
    PRINT RIGHT$(p$, k)
NEXT k
```



RMDIR

(サブディレクトリの削除)

書 式

```
RMDIR path
      ↑ サブディレクト名
```

機 能

path で示されるサブディレクトリを削除します。削除するサブディレクトリ内にはファイルがあってははいけません。

例

```
KILL  "¥SATO¥WORK ¥*.*)" ← サブディレクトリ WORK 内のファイルをすべて削除
RMDIR "¥SATO¥WORK"        ← サブディレクトリ WORK の削除
```

RND

(乱数の発生)

関数

書 式

```
RND [(n)]
      ↑ 乱数発生モード
```

機 能

0 以上 1 未満の単精度実数型の乱数を 1 つ返します。 n の値によって乱数発生モードが異なります。

n	機 能
負	常に同じ値を返す
0	1 つ前の乱数を返す
正または指定せず	次の乱数を返す

n ~ m の整数乱数を発生するためには次の式を用います。

INT((m-n+1) * RND) + n)

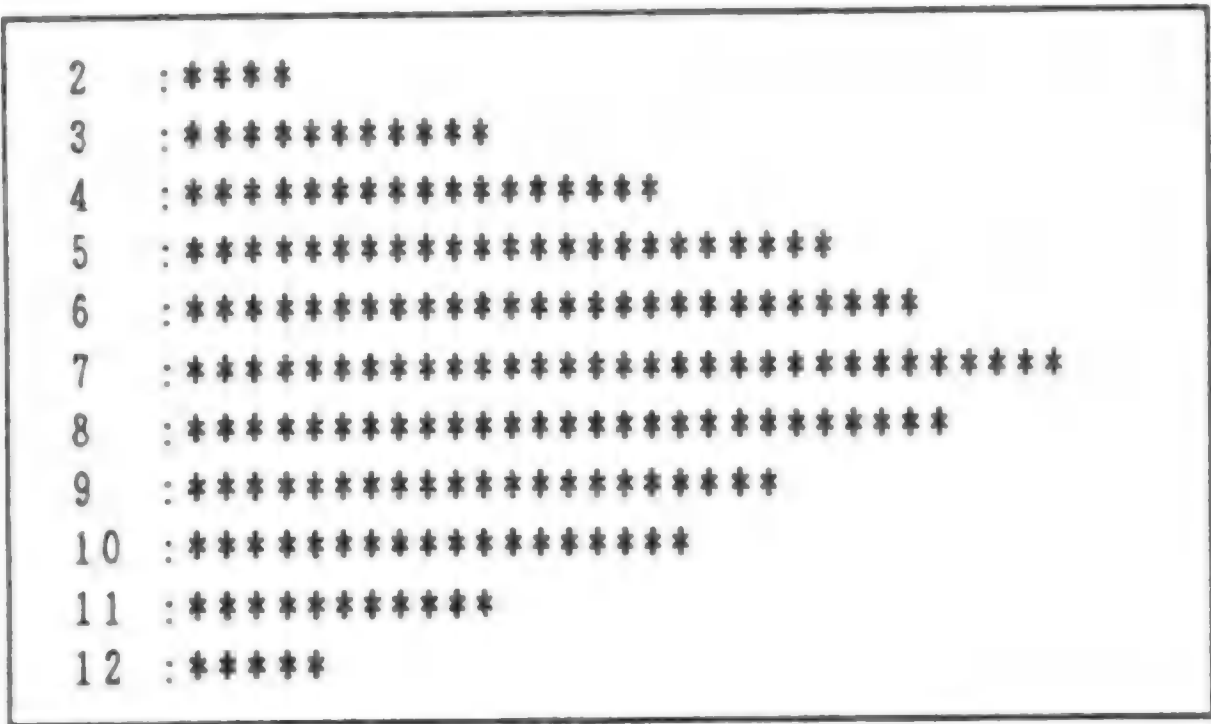
例

```
' ---/* サイコロの目の和の分布 */---

DIM Histo(12)
ERASE Histo

FOR k = 1 TO 4000
  x = INT(6 * RND + 1)    ' 1つ目のサイコロ
  y = INT(6 * RND + 1)    ' 2つ目のサイコロ
  Histo(x + y) = Histo(x + y) + 1
NEXT k

FOR k = 2 TO 12           ' ヒストグラムの表示
  PRINT k; TAB(5); ":";
  FOR j = 1 TO Histo(k) / 20
    PRINT "*";
  NEXT j
  PRINT
NEXT k
```



分布が正規分布になるよう、サイコロをふる回数を4000回としているため、そのままの個数で表示すると画面をはみ出るので20で割っている。

RSET

(ファイルバッファへの値の設定)

書式

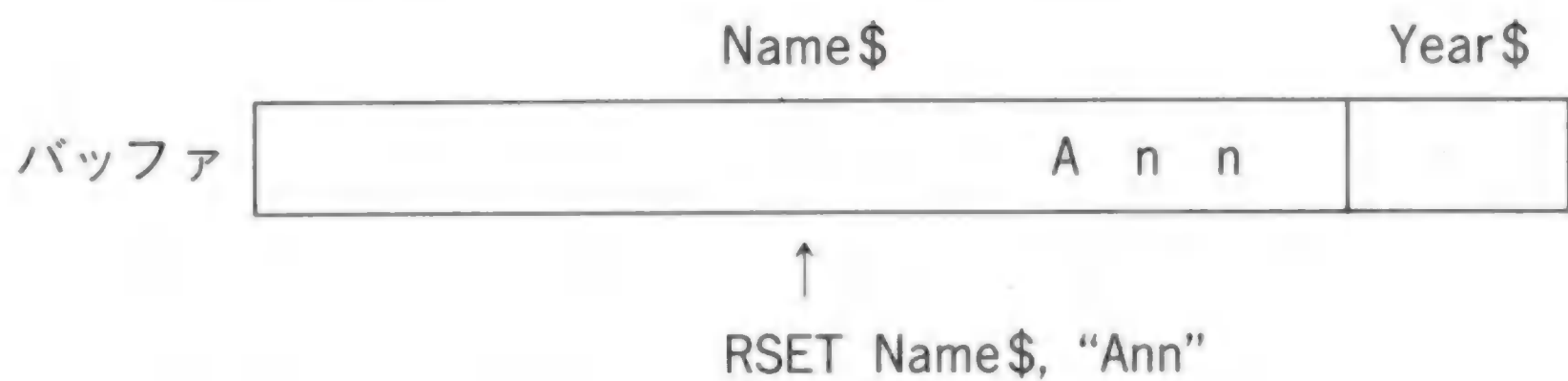
RSET strvar = strexpression

↑ 文字列
↑ FIELD で設定した文字列変数

機能

FIELD 文で設定したリード/ライト・バッファに割り当てられている変数に文字列データを右詰めにして格納します。余った部分は空白で埋められます。文字列が長ければ、はみ出した部分は捨てられます。

FIELD #1, 16 AS Name\$, 2 AS Year\$



strvar には FIELD 文で設定した文字列変数ではなく、一般の固定長文字列変数を使用することができます。機能はフィールド・バッファへの設定と同じです。

なお、LSET では異なるレコード型データ同士の代入が行えましたが、RSET はできません。

例

```
---/* 固定長文字列の右詰め */  
DIM a AS STRING * 10  
  
a = "Hello" ← 左詰め  
PRINT a  
  
RSET a = "Hello" ← 右詰め  
PRINT a
```

Hello
Hello

RTRIM\$

(右側スペースの削除)

関数

書式

RTRIM\$(str)
 ↑ 文字列

機能

文字列 str の右側にあるスペースを削除します。スペースは1バイト文字、2バイト文字のスペースです。

例

```
A$ = "Hello _ BASIC _ _ _ _ _"  
B$ = RTRIM$(A$)  
... "Hello _ BASIC" を返す。
```

RUN

(プログラムの実行)

書式

- ① RUN [line]
 ↑ 実行開始行
- ② RUN filename
 ↑ 実行するプログラム名

機能

- ①メモリ上のプログラムを line で示される行から再実行します。再実行により変数の内容はクリアされます。line にはラベルを使用できません。line を省略するとプログラムの先頭から再実行します。
- ② filename で示されるファイルをメモリ上にロードし実行します。QB 環境下で filename のファイルタイプを省略すると、.BAS とみなされ、DOS 環境下で filename のファイルタイプを省略すると、.EXE とみなされます。
 新しいプログラムのロードにより、前のプログラムは消され、オープンファイルはすべてクローズされます。

例

```
RUN "B: ¥WORK ¥TEST"  
...ドライブ B のサブディレクトリ WORK の TEST.BAS が実行される。
```


SADD

(文字列が格納されているアドレスの取得)

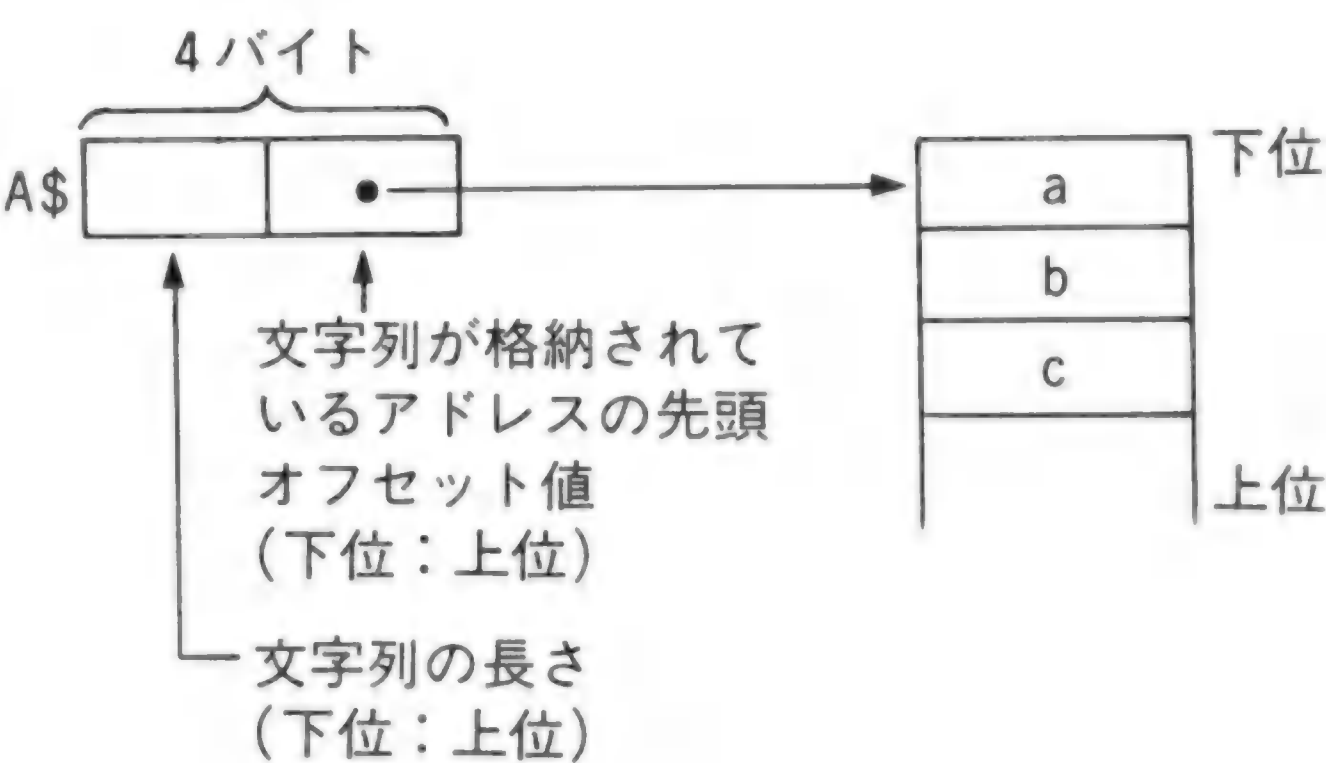
関数

書式

SADD(strvar)
↑ 文字列変数

機能

A\$="abc"のような可変長文字列の格納イメージは、次のようになっています。



つまり、変数 A\$がある場所と、実際に文字列が格納されている場所が異なるわけです。

SADD は、可変長文字列が実際に格納されているメモリ・アドレスの先頭オフセット値を取得します。SADD は固定長文字列には使用できません。

SADD は BASIC 以外の言語で書かれたプロシージャとの間で文字列データを授受する場合によく使います。

例

```
' ---/* 文字列の格納アドレス */---  
DECLARE SUB Disp (adr!)  
  
p$ = "Hello!" + CHR$(0)  
Disp SADD(p$)  
SUB Disp (adr)  
DO  
  a = PEEK(adr)  
  PRINT CHR$(a)  
  adr = adr + 1  
LOOP WHILE a <> 0  
END SUB
```

文字列の終わりの印

先頭アドレス

文字列の終わりまで繰り返す。

H
e
l
l
o
!

… 文字列 p\$ の格納アドレスをプロシージャ Disp に渡し、プロシージャ側では、このアドレスをもとに PEEK で 1 文字ずつ取り出す。文字列の終わりの印としてヌル文字(CHR\$(0))を置くことにする。

SCREEN

(指定位置の文字または色の取得)

関数

書式

SCREEN(y, x[, flag])

↑ 取得モード
↑ 桁位置
↑ 行位置

機能

flag に真を示す値(0 以外の値)を指定すると、テキスト画面の(x, y)位置のカラー属性(属性番号)を返します。

flag に偽を示す値(0)を指定するか、flag を省略すると、テキスト画面の(x, y)位置の文字コードを返します。

例

```
---/* 画面情報の取得 */---  
FOR y = 1 TO 25  
  FOR x = 1 TO 80  
    Col = SCREEN(y, x, 1) ← 色の属性番号を取得  
    Cha = SCREEN(y, x) ← 文字のコードを取得  
    IF Col = 7 THEN  
      LOCATE y, x  
      COLOR 1  
      PRINT CHR$(Cha); ← 青色の文字に変える  
    END IF  
  NEXT x  
NEXT y
```

… テキスト画面の(x, y)位置に表示されている文字の ASCII コードと色番号を取得し、もし色が白ならその文字を青色に変える。

SCREEN

(画面モードの設定)

書式

SCREEN [mode][,[pmode][,[apage][, vpage]]]

↑ ディスプレイページ
↑ アクティブページ
↑ パレット・モード
↑ スクリーンモード

機能

グラフィック画面のモードを設定します。

mode はスクリーンモードで以下の値を指定します。mode を指定して SCREEN 文を実行すると、テキスト画面、グラフィック画面のすべてが消去されます。

mode	機能
81	テキスト
84	640×200 スーパーインポーズ
87	640×400 グラフィック
88, 0	640×400 スーパーインポーズ

pmode はパレット・モードで、使用できる色番号の種類を決めるためのものです。

pmode	機 能
1	16色
2	64色
3	4096色

apage はアクティブページ(描画データを書き込むページ)，vpage はディスプレイページ(スクリーンに表示されるページ)で、スクリーンモードに応じて次の値を指定します。

mode	ページ番号	
	一般の PC-9801	PC-9801U
81	—	—
84	0～3	1
87	0～1	0
88, 0	0～1	0

例

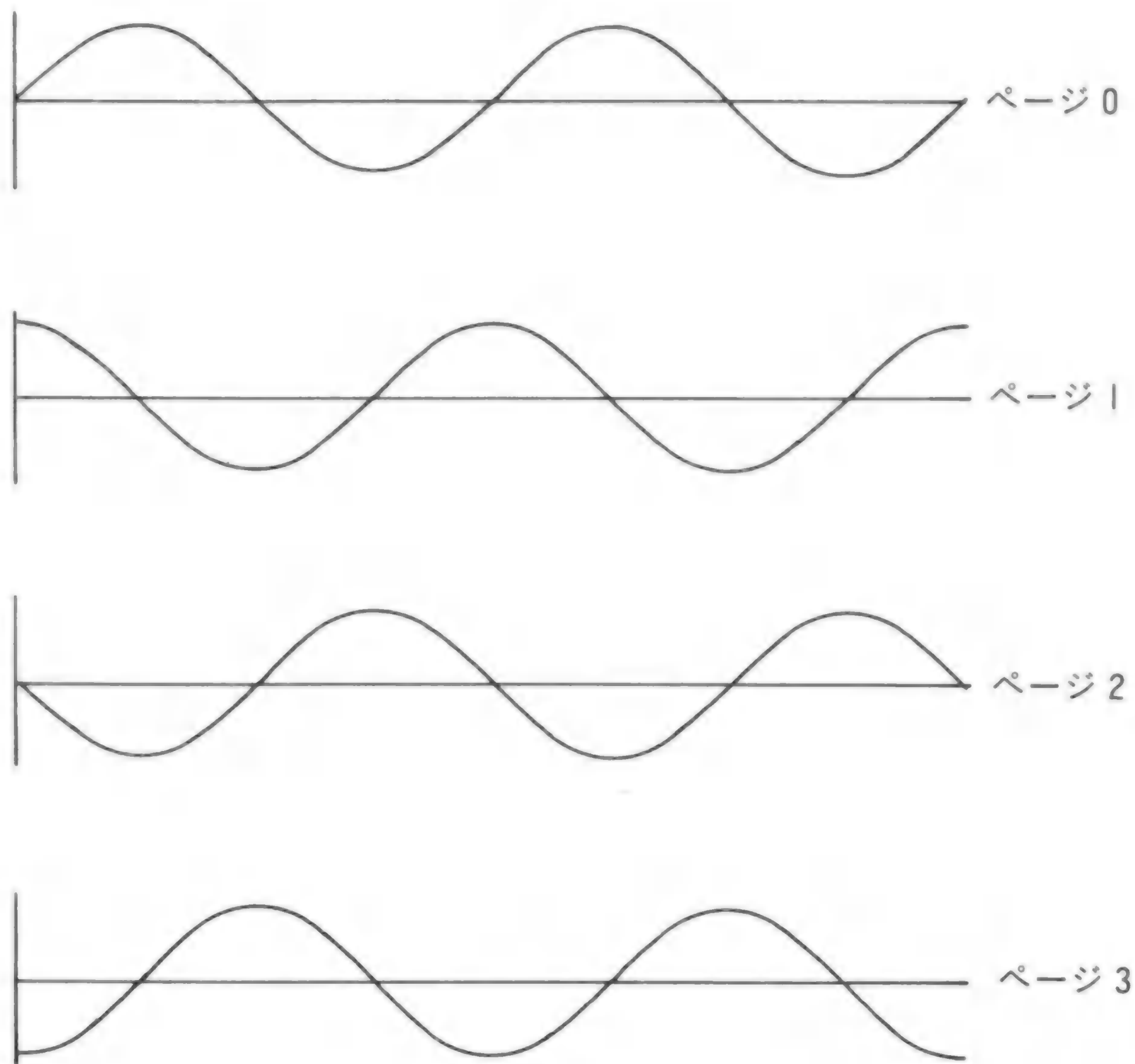
・ アクティブページとディスプレイページ

```
FOR k = 0 TO 3
  SCREEN 84, , k, k ←———— アクティブページ，ディスプレイページの指定
  CLS
  WINDOW (0, -2)-(720, 2)
  LINE (0, -1.2)-(0, 1.2)      ・ 軸線
  LINE (0, 0)-(720, 0)
  FOR x = 0 TO 720 STEP 2
    y = SIN((x + 90 * k) * 3.14159 / 180)
    IF x = 0 THEN
      PSET (x, y)              ・ 最初の点
    ELSE
      LINE -(x, y)
    END IF
  NEXT x
NEXT k

WHILE 1
  FOR k = 0 TO 3
    SCREEN , , , k ←———— ディスプレイページの指定
    FOR t = 1 TO 1000: NEXT t
  NEXT k
WEND
```

←—— サインカーブを描く

… サインカーブの位相を90°ずつずらして4画面に描き、ディスプレイページを逐次切り換えることにより、サインカーブが動いているように見える。



SEEK

(ファイル現在位置の取得)

関数

書式

SEEK(n)
 ↑ ファイル番号

機能

ファイル番号 n のファイルのファイル現在位置を取得します。取得される値はランダム・ファイルにおいてはレコード番号、シーケンシャルファイルにおいては先頭からのバイト数となります。

例

---/* ファイル現在位置 */---

```
OPEN "abc.dat" FOR BINARY AS #1
```

```
DO WHILE NOT EOF(1)
```

```
  p = SEEK(1) ← ファイル現在位置の取得
```

```
  a$ = INPUT$(1, #1) ← 1文字入力
```

```
  IF "a" <= a$ AND a$ <= "z" THEN
```

```
    PRINT p; ":"; a$
```

```
  END IF
```

```
LOOP
```

```
CLOSE #1
```

1	:	a
5	:	a
17	:	t
31	:	u
48	:	a
49	:	b
50	:	c
52	:	d
53	:	a
54	:	t

… ファイルから1文字単位でリードし、その文字が小文字なら、ファイル先頭からの位置を表示する。

SEEK #

(ファイル現在位置の取得)

関数

書 式

SEEK [#]n, pos

↑移動位置

↑ファイル番号

機 能

ファイル番号 n のファイルの、ファイル現在位置を pos 位置に移動します。pos の値は、ランダム・ファイルにおいてはレコード番号、シーケンシャル・ファイルにおいては先頭からのバイト数となります。レコード番号、バイト数の先頭値はともに 1 です。

例

```
---/* ファイル現在位置の移動 */---  
OPEN "abc.dat" FOR INPUT AS #1  
  
SEEK #1, 50  
DO WHILE NOT EOF(1)  
    a$ = INPUT$(1, #1)  
    IF a$ <> CHR$(&HD) THEN PRINT a$;  
LOOP  
CLOSE #1
```

1 文字単位でファイルからリード。
行末にはODH, OAH が置かれているので
2つを出力すると改行が2度行われるため、ODH は出力しない。

… ファイル先頭から50バイトを読み飛ばし、それ以後を1文字ずつ読んでいく。

SELECT CASE

(複数条件判断)

書 式

SELECT CASE test

↑被比較式

CASE p-1

↑比較式

block-1

←実行ブロック

[CASE p-2

block-2]

⋮

[CASE ELSE

block-n]

END SELECT

機 能

test と p-1, p-2 … とを比較し、条件を満たしたところの block を実行し、SELECT CASE 文から抜けます。もし条件を満たさなければ、CASE ELSE 節に置かれている block-n を実行します。
block には複数の文を置くことができます。

test には変数を含む数式または文字列式を書きます. p-1,p-2, ...には次の書式のいずれかを書きます. 1つのCASE節にこれらの比較式をカンマ(,)で区切って複数指定することもできます.

式1, 式2, ...
式1 TO 式2
IS { < <= > >= <> = } 式

例

```
INPUT x
SELECT CASE x
  CASE 1
    xが1のときの処理
  CASE 2 TO 5
    xが2~5のときの処理
  CASE 6, 8, 9
    xが6か8か9のときの処理
  CASE IS>=10
    xが10以上のときの処理
END SELECT
```

左の値は右の値より小さく
なければいけない.
カンマ(,)で複数の式を並
べて書ける. この場合は
OR条件と考えてよい.
x IS>=10という意味に
考えればよい.

```
INPUT Test$
SELECT CASE Test$
  CASE "A" TO "AZZZZZZZZZ"
    Test$がAで始まる文字のときの処理
  CASE IS < "A"
    Test$がアルファベット以外の文字のときの処理
  CASE "Quick"
    Test$が"Quick"のときの処理
END SELECT
```


SETMEM (far ヒープで使用するメモリ量の設定) 関数

書 式 SETMEM(n)
 ↑ 増減するバイト数

機 能

far ヒープメモリのサイズを n バイト増減します。n に負の値を用いると far ヒープのメモリサイズを減少させます。

BASIC では、プログラム起動時に far ヒープは最大にとられているので、最初から SETMEM で far ヒープを拡大することはできません。

SETMEM は増減後の far ヒープのバイト数を返します。したがって n に 0 を指定すると、現在の far ヒープのバイト数がわかります。

例

```
PRINT "現在のヒープ・サイズ="; SETMEM(0)
a = SETMEM(-1024)
PRINT "変更後のヒープ・サイズ="; a
```

現在のヒープ・サイズ= 178720
変更後のヒープ・サイズ= 177696

SGN (正・負符号の取得) 関数

書 式 SGN(x)

機 能

$x > 0$ のとき 1, $x = 0$ のとき 0, $x < 0$ のとき -1 をそれぞれ返します。

例

```
D = B * B - 4 * A * C
ON SGN(D) + 2 GOSUB KYOKON, JYUKON, JITSUKON
```

… 判別式 D の符号に応じて KYOKON(虚根), JYUKON(重根), JITSUKON(実根) へ分岐する。

SHARED

(プロシージャ間の変数の共有)

書 式

SHARED var [AS type], ...

↑ 変数 ↑ 変数の型

機 能

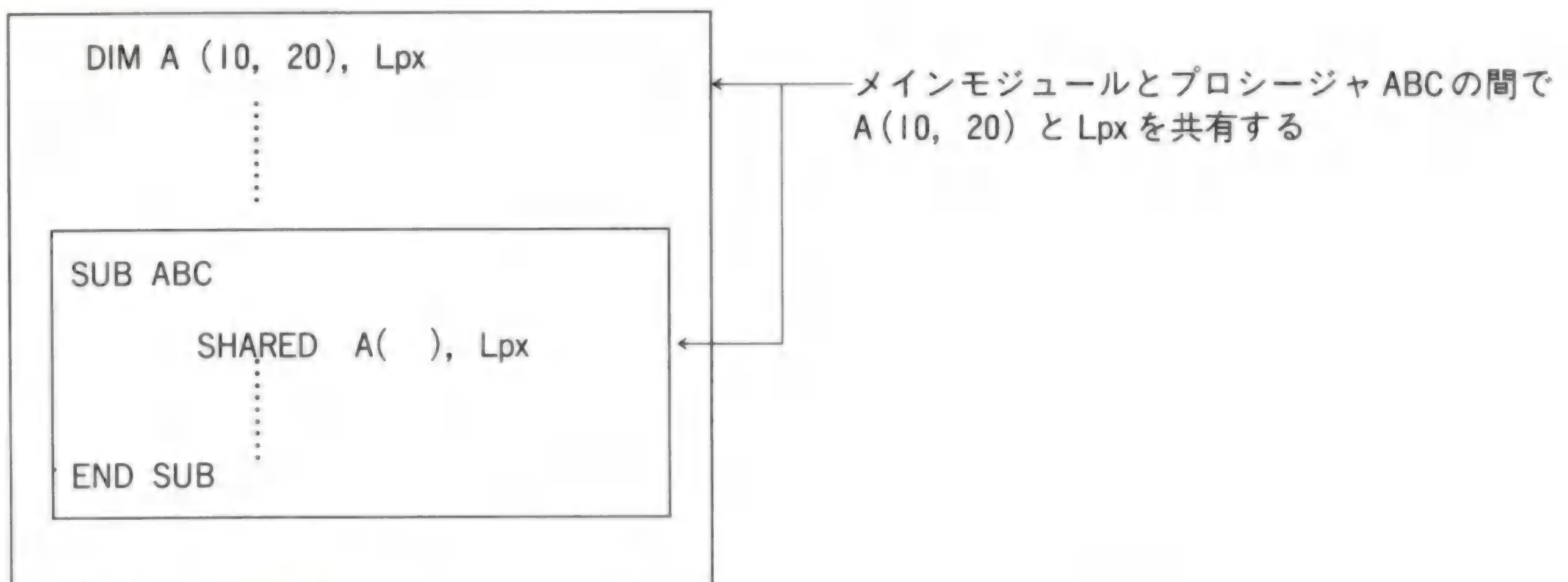
メインモジュールで宣言または使用されている変数を、SHARED 文を行なったプロシージャで共有します。配列の場合は、() を付けます。

SHARED 文を置くことができるのはプロシージャの中だけです。

COMMON 文または DIM 文に SHARED 属性を指定すれば、その変数は全プロシージャで共有されますが、SHARED 文で指定した変数は、その SHARED 文が置かれているプロシージャだけで共有されます。

別個にコンパイルしてリンクしたプロシージャとの間では SHARED 文は効果がありません。

例



SHELL (実行中のプログラムを一時中断して他のプログラムを実行)

書 式

SHELL [commandstring]

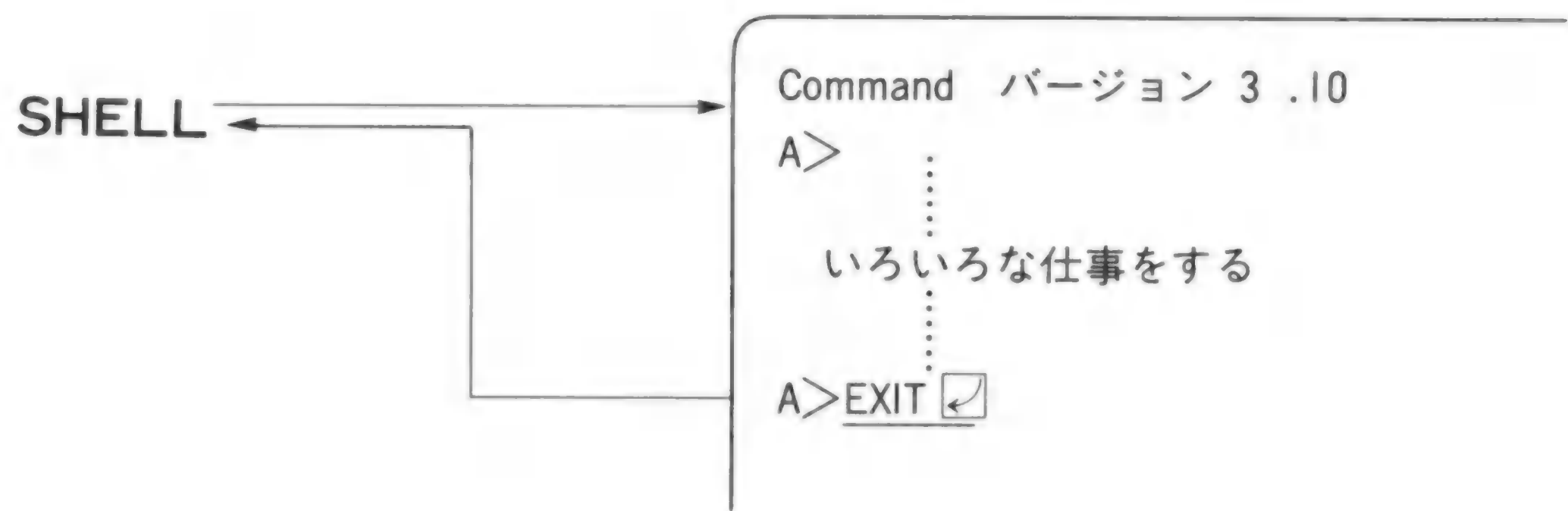
↑ 実行するプログラムのコマンドライン文字列

機 能

BASIC プログラム内からほかのプログラム(これを子プロセスと呼ぶ)を実行し、実行終了後、SHELL 文の次の文に戻ります。

commandstring は実行プログラム名と、それに渡されるパラメータからなる文字列です。実行プログラムのファイルタイプを省略すると、.COM, .EXE, .BAT の順で探します。

commandstring を省略すると、COMMAND.COM が起動されます。この場合、プログラムに戻るには A>EXIT ☒ とします。



例

```
SHELL "TYPE B: ABC.BAS"
...TYPE コマンドを実行する.
SHELL "DIR B:* .BAS > DIRFILE"
...DIR をとった結果をI/O リダイレクトして、ファイル DIRFILE に書き込む.
```

SIN

(サイン)

関数

書式

SIN(x)

機能

x のサインの値を求めます。x の単位はラジアンです。π(3.14159)ラジアンが180°に相当しますから、A°は A * 3.14159/180 [ラジアン] となります。

例

```
' ---/* 三角関数 */---
rd = 3.14159265# / 180
PRINT "      x"; TAB(23); "sin(x)"; TAB(43); "cos(x)"; TAB(63); "tan(x)"
FOR x = 0 TO 90 STEP 10
  PRINT USING "####.## "; x;
  PRINT USING "#####.#####"; SIN(x * rd); COS(x * rd); TAN(x * rd)
NEXT x
```

x	sin(x)	cos(x)	tan(x)
0.0	0.000000	1.000000	0.000000
10.0	0.173648	0.984808	0.176327
20.0	0.342020	0.939693	0.363970
30.0	0.500000	0.866025	0.577350
40.0	0.642788	0.766044	0.839100
50.0	0.766044	0.642788	1.191754
60.0	0.866025	0.500000	1.732051
70.0	0.939693	0.342020	2.747477
80.0	0.984808	0.173648	5.671280
90.0	1.000000	-0.000000	-22877334.000000

SPACE\$

(スペースの生成)

関数

書 式

SPACE\$(n)
↑ スペースの数

機 能

n 個のスペースからなる文字列を生成します。

例

’ ---/* 任意の大きさの四角を描く */---

INPUT "縦,横 "; m, n

PRINT STRING\$(n, "=") ←=====を n 個作る

FOR k = 1 TO m

PRINT "I" + SPACE\$(n - 2) + "I"

NEXT k

PRINT STRING\$(n, "=")

```

縦,横 ? 5,8
=====
I           I
I           I
I           I
I           I
I           I
=====
    
```

SPC

(スペースの出力)

関数

書 式

SPC(n)
↑ スペースの数

機 能

PRINT または LPRINT 文と共に使用し、スクリーンまたはプリンタに n 個のスペースを出力します。

例

PRINT "....+.+.+.+...."

PRINT "Hello"; SPC(5); "Basic"

```

....+. ....+. ....+. ....+....
Hello      Basic
    
```

SQR

(ルート：平方根)

関数

書 式

SQR(x)

機 能

\sqrt{x} を求めます。x は正の値を指定してください。

STATIC**(静的変数の宣言)**

書 式 **STATIC** var [AS type], ...
 ↑変数 ↑変数の型

機 能

変数を静的変数として宣言します。静的変数は、それが宣言されているプロシージャまたは DEF FN 関数でローカルです。DEF FN 関数内の変数はデフォルトでグローバルです。これをローカルにしたいときに STATIC 宣言を行います。また、自動変数のようにプロシージャのコール/リターンのたびにスタック上に生成/消滅をくり返さず、メモリ上の同じ領域にプログラムの開始から終了まで割り当てられていますから、変数の値は消滅せず保存されています。

STATIC 文を置くことができるのは、プロシージャと DEF FN 関数の中だけです。

SUB 文、FUNCTION 文に STATIC 属性を指定すると、プロシージャ内のすべての変数は静的変数となりますが、SHARED 宣言されている変数はこの限りではありません。これに対し、STATIC 文は指定した変数を静的変数にするもので、SHARED 宣言されている共有変数を無効にすることもできます。

STOP**(実行の停止)**

書 式 **STOP**

機 能

プログラムの実行を停止します。

QB 環境下で STOP 文が実行されると、QB 画面に戻ります。 .EXE ファイル中の STOP 文が実行されると、すべてのファイルをクローズして OS に戻ります。

コンパイラ BC で /d, /e, /x オプションを指定していると、STOP 文が実行された行番号が表示されます。STOP 文に行番号が付いていなければ最も近い位置の行番号が表示され、行番号がどこにもなければ 0 が表示されます。

STR\$

(数値を10進文字列に変換)

関数

書 式

STR\$(n)
↑ 数値

機 能

数値 n を10進文字列に変換します。正の値なら先頭に空白が1つ付けられます。

例

```
PRINT STR$(123)
PRINT STR$(-123)
PRINT STR$(100.05)
PRINT STR$(&HFF)
```

```
123
-123
100.05
255
```

STRING\$

(文字列の生成)

関数

書 式

- ① STRING\$(n, str)
↑ 文字列
↑ くり返し回数
- ② STRING\$(n, numeric)
↑ ASCIIコードまたはシフト JISコード
↑ くり返し回数

機 能

- ① 文字列 str の先頭文字(1バイト文字または2バイト文字)を n 個くり返した文字列を返します。
- ② numeric で示される ASCII コード, またはシフト JIS コードに対応した文字を n 個くり返した文字列を返します。

例

SPACE\$参照。

SUB

(サブルーチン・プロシージャの定義)

書 式

```
SUB name[(var [AS type],...)] [STATIC]
:
↑ 仮引数の型
↑ 仮引数
↑ プロシージャ名
END SUB
```


機能

name で示されるサブルーチン・プロシージャを定義します。() の中に仮引数を書きます。仮引数がない場合は() を省略できます。

STATIC を指定すると、関数プロシージャ内のローカル変数はすべて (SHARED 宣言されている変数は除く) 静的変数となります。静的変数については第4章4-2の8を参照。

例

```
' ---/* サブルーチン・プロシージャ */---
```

```
DECLARE SUB Disp (Moji$, Num!, Col!) ← サブルーチン・プロシージャの宣言
```

```
CLS
Disp "Hello Quick Basic", 2, 3
Disp "Hello Quick C", 3, 5
```

↑ くり返し回数
↑ 表示色

```
END
```

```
SUB Disp (Moji$, Num, Col)
  COLOR Col
  FOR k = 1 TO Num
    PRINT Moji$
  NEXT k
END SUB
```

↑ サブルーチン・プロシージャの定義。
Moji\$の文字列を Num 回、色 Col で表示。

SWAP

(変数の内容を交換)

書式

```
SWAP var1, var2
      ↑   ↑
      交換する変数
```

機能

2つの変数 var1と var2のデータを交換します。2つの変数の型は一致していなければなりません。

例

```
A = 10 : B = 20
SWAP A,B
```

…A が20, B が10となる。

SYSTEM

(プログラムを終了し OS に戻る)

書式

```
SYSTEM
```

機能

QB 環境下で SYSTEM 文が実行されると QB 画面に戻ります。ただし、QB 起動時に /run オプションが指定されていた場合は、SYSTEM 文が実行されると OS に戻ります。

.EXE ファイルの中で SYSTEM 文が実行されると、すべてのファイルをクローズして OS に戻ります。

ダイレクトモードで SYSTEM コマンドを入力すると、QB を終了して OS に戻ります。

TAB

(表示位置の指定)

関数

書式

TAB(n)

↑ 画面左端からの桁位置

機能

PRINT, LPRINT 文と共に用い、スクリーンまたはプリンタの左端(桁位置は 1)からの表示/印字位置を n に設定します。

現在の表示/印字位置より小さい n の値を指定すると、1 行改行されて n 桁目に表示/印字されます。

n の値が 1 行の幅より大きい場合は、次の行に移り、(n MOD 幅)の位置に表示/印字されます。

n の値に 1 より小さい値を指定すると、1 とみなされます。

例

' ---/* サイン・カーブ */---

rd = 3.14159 / 180

PRINT TAB(19); "-1 0 1"

PRINT TAB(20); "1....+....1....+....1"

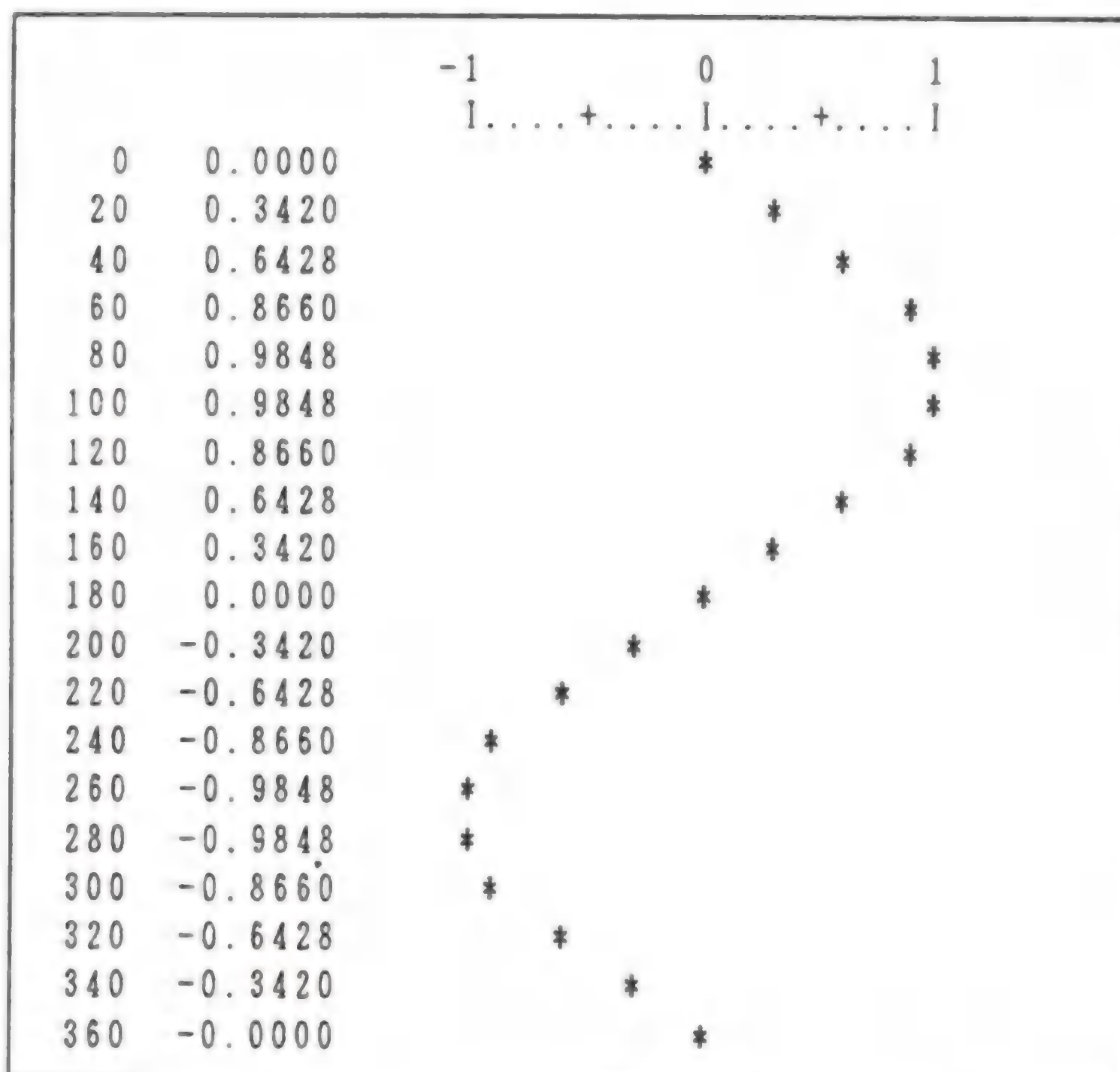
FOR x = 0 TO 360 STEP 20

ts = 30 + 10 * SIN(x * rd) ← 表示位置

PRINT USING "##### ###.####"; x; SIN(x * rd);

PRINT TAB(ts); "*"

NEXT x



TAN (タンジェント)		関数
書 式	TAN(x)	
機 能	x のタンジェントを求めます。x の単位はラジアン。	
例	SIN 参照。	

TIMES\$ (現在の時刻の取得)		関数
書 式	TIMES\$	
機 能	現在の時刻を、hh:mm:ss の 8 文字の文字列で取得します。 hh : 時(00~23) mm : 分(00~59) ss : 秒(00~59)	
例	<pre>---/* 時刻の取得と設定 */--- CLS PRINT "現在の時刻-->"; TIMES\$ ← 時刻の取得 INPUT "時.分.秒 "; hour\$, minute\$, second\$ TIMES\$ = hour\$ + ":" + minute\$ + ":" + second\$ ← 時刻の設定</pre> <div>現在の時刻-->15:32:16 時.分.秒 ? 16.10.00</div>	

TIMES\$ (時刻の設定)	
書 式	TIMES\$ = str ↑時刻を示す文字列
機 能	str で示す時刻に現在時刻を設定します。str は hh:mm:ss の文字列です。mm:ss または ss を省略すると、省略されたものは00に設定されます。 hh : 時(00~23) mm : 分(00~59) ss : 秒(00~59)

TIMER

(午前 0 時からの経過秒の取得)

関数

書 式

TIMER

機 能

午前 0 時からの経過秒を取得します。

例

```
st& = TIMER  
PRINT "これから 3 0 秒間眠ります。 . .  
WHILE TIMER - st& < 30  
WEND
```

...30秒間経過するまで WHILE～WEND ループをまわる。

TIMER ON/OFF/STOP

(タイマイイベント・トラッピングの開始/禁止/停止)

書 式

- ① TIMER ON
- ② TIMER OFF
- ③ TIMER STOP

機 能

- ①タイマイイベント・トラッピングの開始
- ②タイマイイベント・トラッピングの禁止
- ③タイマイイベント・トラッピングの停止。停止中に発生したトラッピングは記憶されており、トラッピングが許可(開始)されると、ただちにイベント処理ルーチンに分岐します。

例

ON event GOSUB 参照。

TRON/TROFF(トレース・オン/トレース・オフ)

書 式 ① TRON
② TROFF

機 能 ①トレースモードをオンにします。つまり、1行ごと実行しながら実行
行を反転表示します。
②トレースモードをオフにします。

TRON は、D/デバッグメニューの T/トレースを指定したのと同じ
効果が得られます。

例

⋮
TRON

TROFF
⋮

← この範囲が実行されるときは、実行行が1行ごと
反転表示される。

TYPE (レコード型の定義)

書 式

↓レコード型に付けた名前

TYPE tag
member AS type
member AS type
⋮
↑メンバの型
↑メンバ名
END TYPE

機 能 tag という名のレコード型を定義し、そのレコードを構成するメンバ
のメンバ名とメンバの型を定義します。
メンバの型には、INTEGER, LONG, SINGLE, DOUBLE, 固定長
文字列型, ユーザ定義型(レコード型)を指定できます。可変長文字列は
メンバの型としては使えないことに注意してください。
レコード型変数は、tag を用いて、DIM, REDIM, COMMON,
STATIC, SHARED 文で行います。
レコードの各メンバの参照は、ピリオド(.) を用いて、
レコード型変数. メンバ名
で行います。
レコード型の定義はプロシージャの中では行えません。

例

```
' ---/* 木星の4大衛星 */---  
  
TYPE Eisei                      ' E i s e i 型の定義  
    Star AS STRING * 10  
    Kodo AS INTEGER  
    Shuki AS SINGLE  
END TYPE  
  
DIM a(4) AS Eisei                ' E i s e i 型変数の宣言  
DIM b(4) AS Eisei  
  
FOR k = 1 TO 4  
    READ a(k).Star, a(k).Kodo, a(k).Shuki  
NEXT k                            ↑ メンバの参照  
  
FOR k = 1 TO 4  
    b(k) = a(k)                  ' レコード・データの一括代入  
    PRINT b(k).Star, b(k).Kodo, b(k).Shuki  
NEXT k  
  
DATA Io, 5, 1.7691, Europa, 6, 3.5512  
DATA Ganymede, 5, 7.1545, Callisto, 6, 16.6890
```

Io	5	1.7691
Europa	6	3.5512
Ganymede	5	7.1545
Callisto	6	16.689

UBOUND

(配列添字の上限値の取得)

関数

書 式

```
UBOUND(array[,d])  
      ↑      ↑  
      配列名 添字の次元
```

機 能

配列 array の d 次の添字の上限値を取得します。
LBOUND, UBOUND は引数渡しされた配列のサイズを取得するのに使います。

```
DIM A(-10 TO 20, 5 TO 10)
```

```
UBOUND(A, 1) ... 20を返す。  
UBOUND(A, 2) ... 10を返す。
```

例

LBOUND 参照。

UCASE\$

(大文字に変換)

関数

書 式	UCASE\$(str) ↑ 文字列
機 能	文字列中の小文字を大文字に変換して返します。対象となる文字は1バイト文字、2バイト文字のa～zです。
例	INPUT A\$ IF UCASE\$(A\$)= "YES" THEN～ …A\$に Yes, yes などが入力されても YES として比較できる。

UNLOCK (他のプロセスからのアクセス禁止を解除)

☞ LOCK 参照。

VAL

(文字列を数値に変換)

関数

書 式	VAL(str) ↑ 文字列
機 能	文字列 str を数値に変換します。先頭に&h または&H があれば16進数とみなされます。文字列の先頭にある空白、タブ、ラインフィードは読み飛ばされますが、文字列の途中にあってはいけません。もし、数字文字でない文字があると、その直前までの文字列を数値に変換します。最初の文字が数字文字でない場合は0 を返します。
例	VAL("123") … 123 を返す。 VAL("&HFF") … 255 を返す。 VAL("FF") … 0 を返す。 ' ---/* 文字列を数値に変換 */--- FOR k = 1 TO 5 READ a\$ PRINT VAL("&h" + a\$) NEXT k DATA ff, 1a, 1b, 80, 7f … DATA に置かれている16進文字列データを数値に変換する。

VARPTR/VARSEG

(変数のアドレスのオフセット値/セグメント値の取得)

関数

書 式

VARPTR(var)

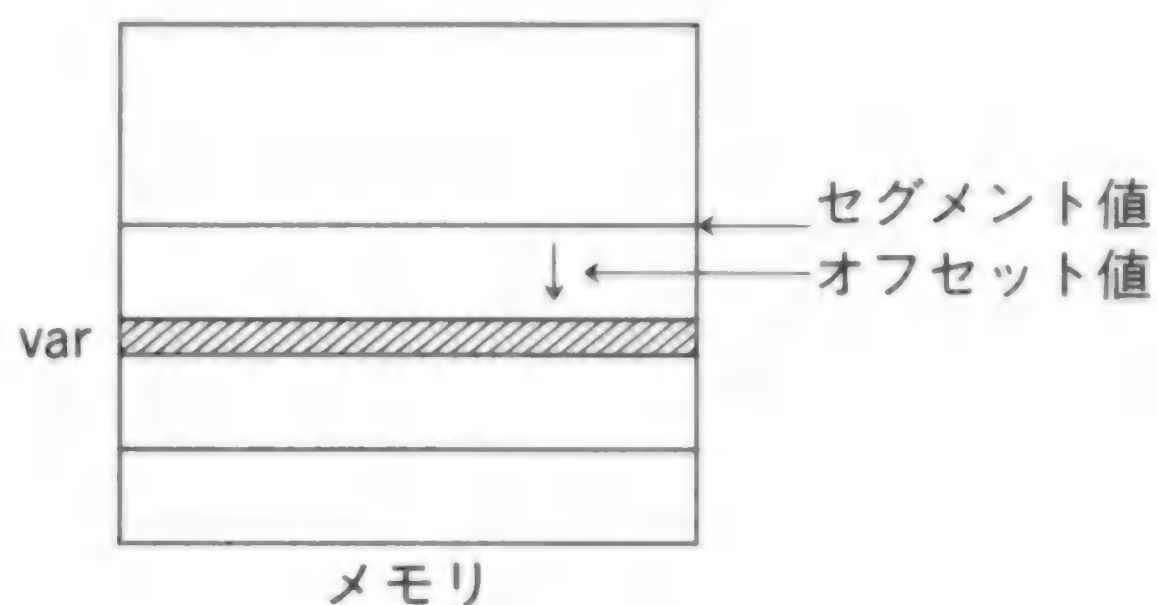
VARSEG(var)

↑ 変数名

機 能

変数 var が割り当てられているメモリ上のアドレスのオフセット値とセグメント値を取得します。

配列のアドレスを取得する場合は、var として配列の第 1 要素を指定してください。



例

’ ---/* 変数への直接アクセス */---

CONST N = 100

DIM a%(N)

segment = VARSEG(a%(0))

offset = VARPTR(a%(0))

DEF SEG = segment

FOR adr = offset TO offset + N * 2 STEP 2

POKE adr, &H1

POKE adr + 1, &H0

NEXT adr

FOR k = 0 TO N

PRINT a%(k);

NEXT k

…配列の先頭アドレスを取得し、POKE を用いてそのアドレスを直接アクセスする。

← セグメントの設定

← 下位バイト

← 上位バイト

VARPTR\$ (変数のアドレスを文字列として取得) 関数

書式

VARPTR\$(var)

↑変数名

機 能 変数 var が割り当てられているメモリ上のアドレスのオフセット値を文字列として取得します。

VARPTR\$は、DRAW の引数に用いる変数のアドレスを取得するのに使われます。

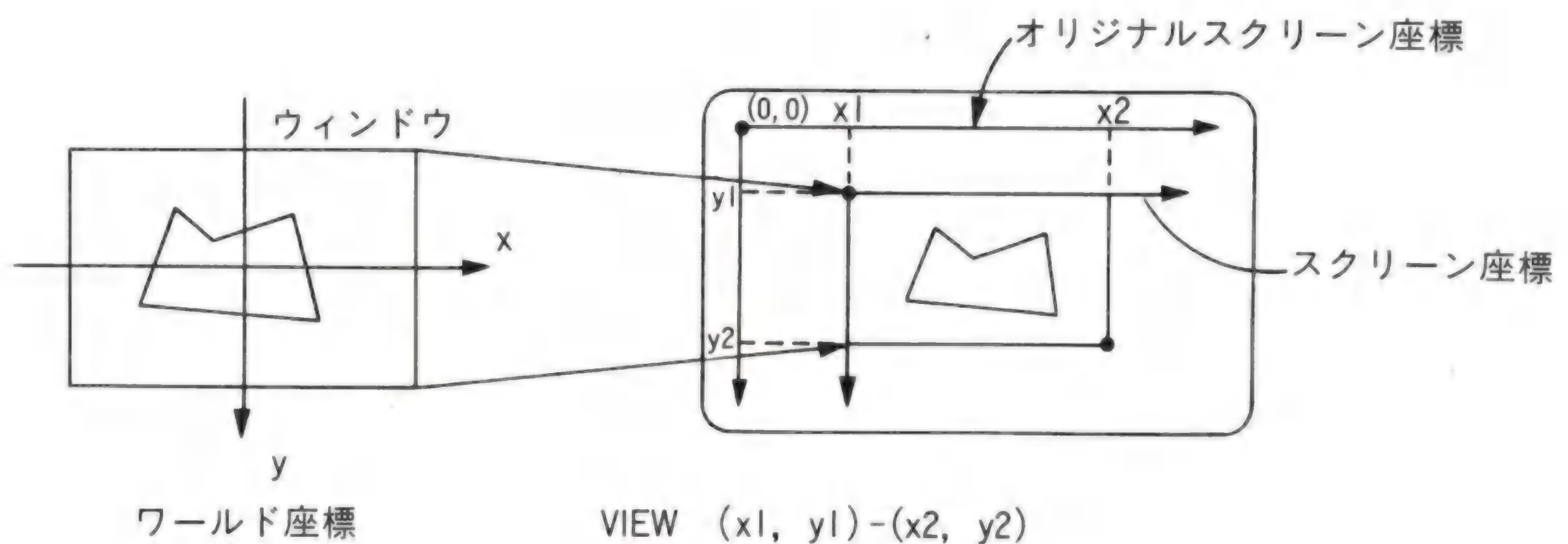
例 DRAW 参照.

VIEW (ビューポートの設定)

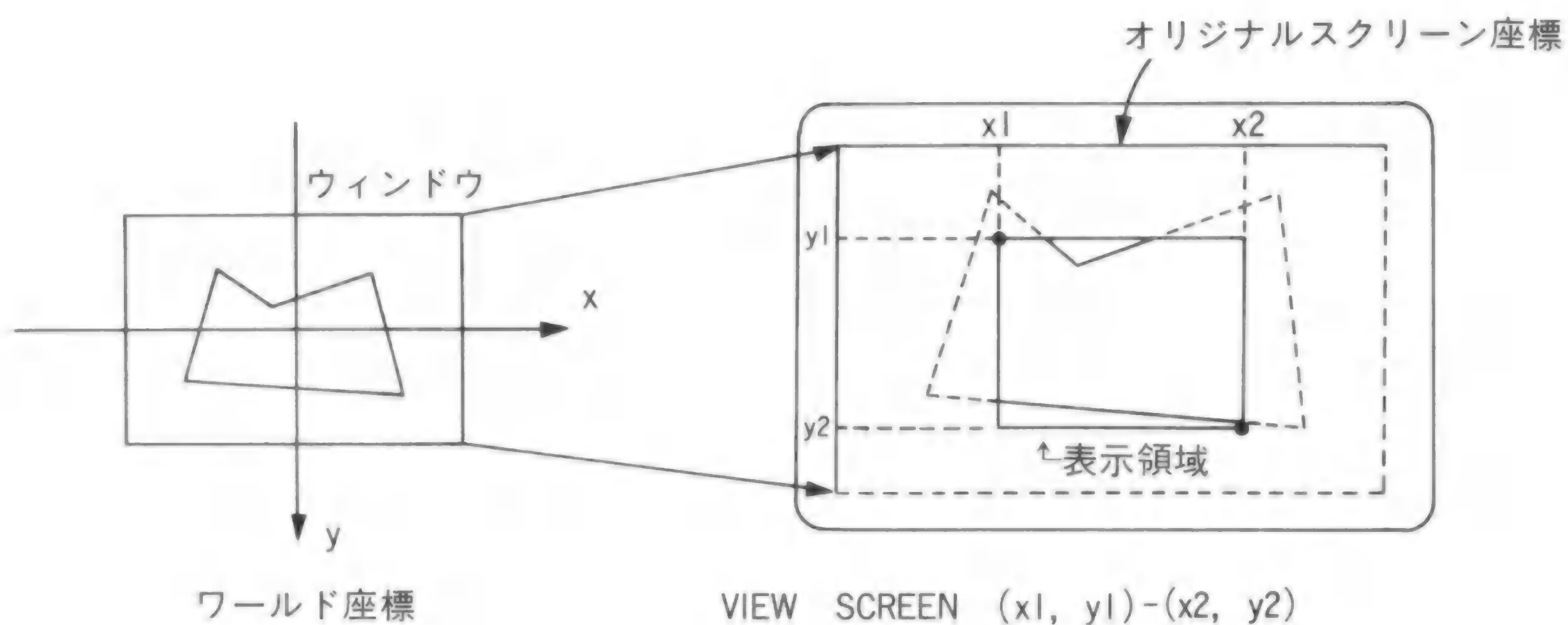
書式 VIEW [[SCREEN](x1, y1) - (x2, y2)[,[color][,border]]]

↑ビューポートの境界線の色
 ↑ビューポート内を塗る色
 ↑ビューポートを示す範囲

機 能 VIEW (x1, y1) - (x2, y2)はオリジナルスクリーン座標上の (x1, y1) - (x2, y2)をビューポートに設定し、その領域に WINDOW で指定された領域を表示します。



これに対し、VIEW SCREEN(x1, y1)-(x2, y2)は、WINDOWで指定された領域をオリジナルスクリーン座標上((0,0)-(639, 399))に表示しますが、実際に表示する範囲を(x1, y1) - (x2, y2)の範囲に限定します。



color はビューポート内を塗る色, border はビューポートを示す四角形の枠を描く色です.

引数を省略し, 単に VIEW とした場合は, 画面全体がビューポートとなります.

例

---/* ビューポートの設定 */---

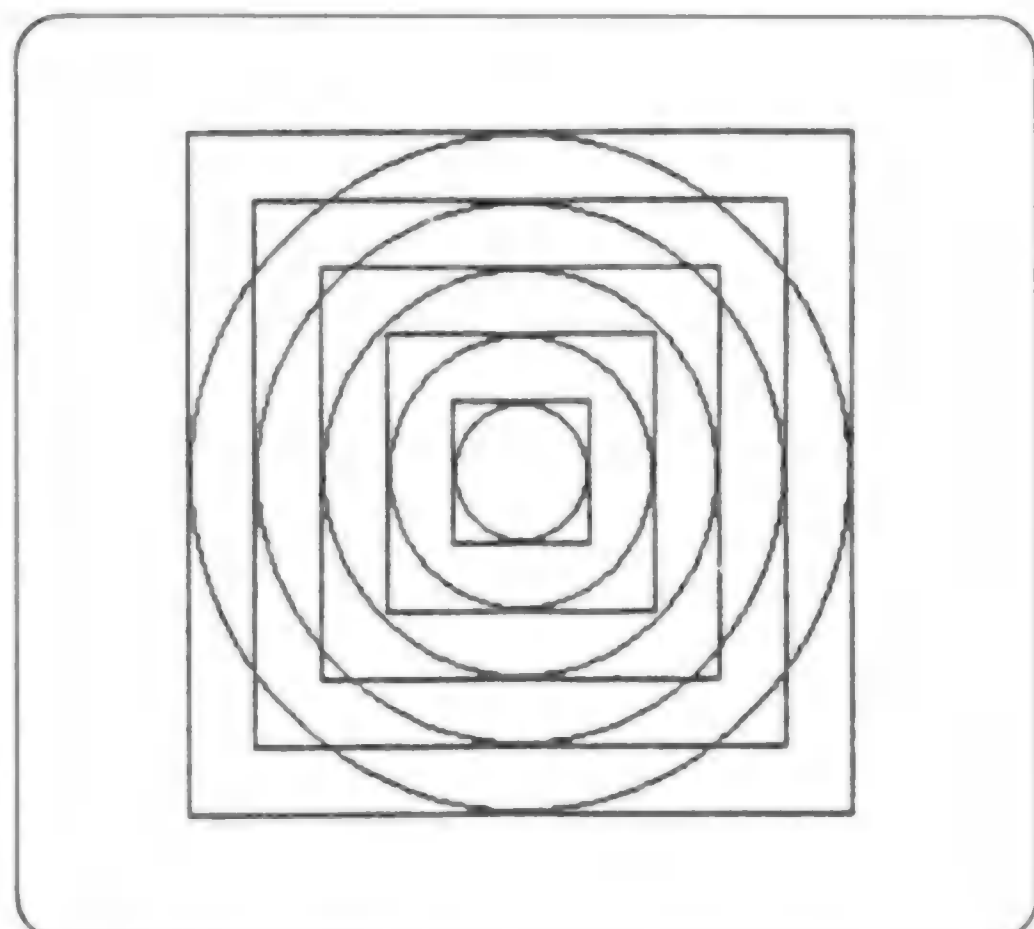
```
SCREEN 0: CLS
WINDOW (-50, -50)-(50, 50)
```

```
x1 = 1: y1 = 1
x2 = 201: y2 = 201
```

```
FOR k = 0 TO 4
    VIEW (x1, y1)-(x2, y2), , 1
    CIRCLE (0, 0), 50
    x1 = x1 + 20: y1 = y1 + 20
    x2 = x2 - 20: y2 = y2 - 20
NEXT k
```

ビューポート枠を青色で表示

ビューポートを狭くしていく



VIEW PRINT (テキスト・ビューポートの上下行の指定)

書 式

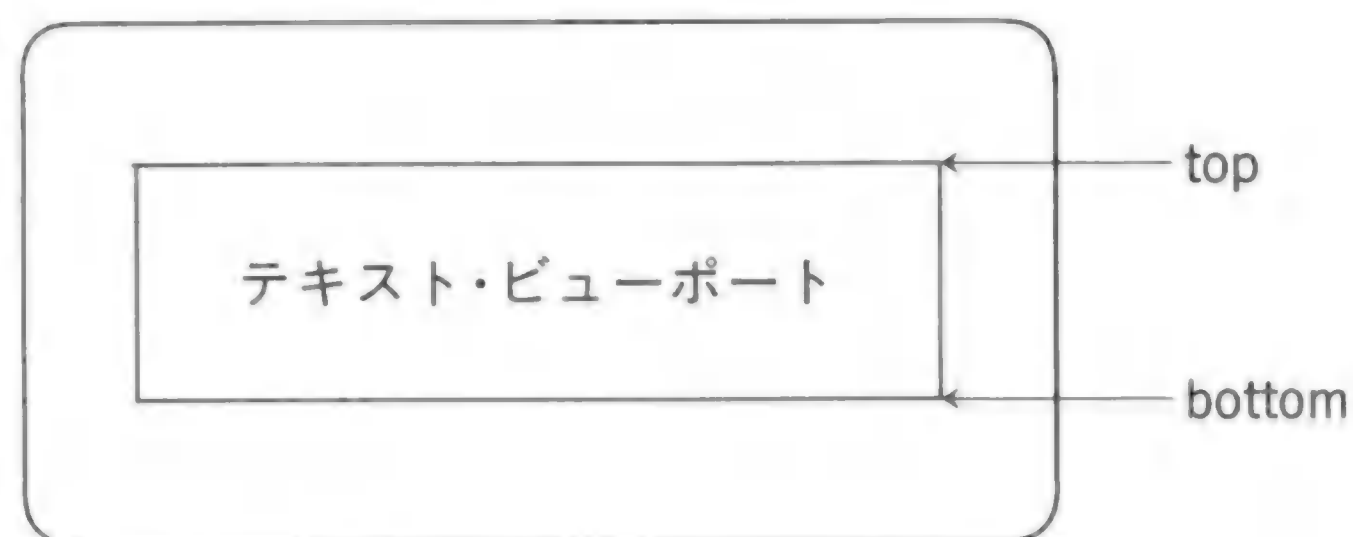
VIEW PRINT [top TO bottom]

↑ 最上行

↑ 最下行

機 能

テキスト・ビューポートの上下行を指定します。



画面の最上行は1行からです。

引数を省略すると画面全体がテキスト・ビューポートになります。

例

```
---/* テキスト・ビューポートの設定 */---
```

```
VIEW PRINT 5 TO 15
```

← テキスト・ビューポートを5行～15行に設定。この範囲が表示域となる。

```
FOR k = 1 TO 100
  PRINT k, "Hello"
NEXT k
```

WAIT

(入力ポートの状態を調べるあいだプログラムを停止)

書 式

WAIT n, and-expression[, xor-expression]

↑

↑ XOR を行う整数式

↑ AND を行う整数式

↑ ポート番号

機 能

ポート n からデータを読み、そのデータと xor-expression との XOR をとり、さらにその値と and-expression との AND をとります。そしてその値が 0 なら再びポートからデータを読み同じことを繰り返します。値が 0 でなくなったときに WAIT 文の次の文に進みます。xor-expression は省略することができ、そのときは、AND 演算から行われます。

例

· ---/* プリンタ・ポートへの1文字出力 */---

DECLARE SUB lputc (c%)

lputc (ASC("a"))

lputc (ASC("b"))

lputc &HD: lputc &HA 改行

SUB lputc (c%)

WAIT &H42, 4 ← ポート&H42のデータの第2ビットが
OUT &H40, c% 1になるまで待つ。

OUT &H46, 14

OUT &H46, 15

END SUB

…lputc(c%)はASCIIコードc%に対応する文字を
プリンタに出力する。

WHILE~WEND

(前判定反復)

書式

WHILE condition

↑ 条件式

WEND

機能

条件式 condition が真のあいだ、ループをくり返します。WHILE
~WEND は DO WHILE~LOOP と同じです。ただし、DO~LOOP
は DO EXIT によりループ外に脱出できますが、WHILE~WEND で
は GOTO で脱出するしかありません。

例

· ---/* 前判定反復 */---

Sum = 0

INPUT a

WHILE a <> -9999

Sum = Sum + a

INPUT a

WEND

PRINT "合計="; Sum

WIDTH

(画面幅の設定)

書 式

WIDTH [column][,line]

└─ 桁数

└─ 行数

機 能

テキスト画面のサイズを設定します。Ver.4.2ではcolumnに80, lineに25以外の値を指定することはできません。Ver.4.5では, lineに20も指定できます。

WIDTH (ファイルおよびデバイスの1行の長さの設定)

書 式

① WIDTH #n, width

└─ 1行の長さ

└─ ファイル番号

② WIDTH device, width

└─ 1行の長さ

└─ デバイス・ファイル名

③ WIDTH LPRINT width

└─ 1行の長さ

機 能

①ファイル番号nでオープンされているデバイス・ファイルの1行の長さをwidthに設定します。

②deviceで示されるデバイス("cons:"など)の1行の長さをwidthに設定します。この命令が効果を現すのは、このデバイスが次のOPEN文でオープンされてからあとです。

③以後のLPRINT文を使用するラインプリンタの1行の長さをwidthに設定します。

例

· ---/* プリンタの出力幅の設定 */---

a\$ = "abcdefghijklmnopqrstuvwxy"

WIDTH LPRINT 5

LPRINT a\$

WIDTH LPRINT 10

LPRINT a\$

abcde

fg hij

klmno

pqrst

uvwxy

z

abcdefghijklmnop

klmnopqrst

uvwxyz

WINDOW

(ウィンドウの設定)

書 式

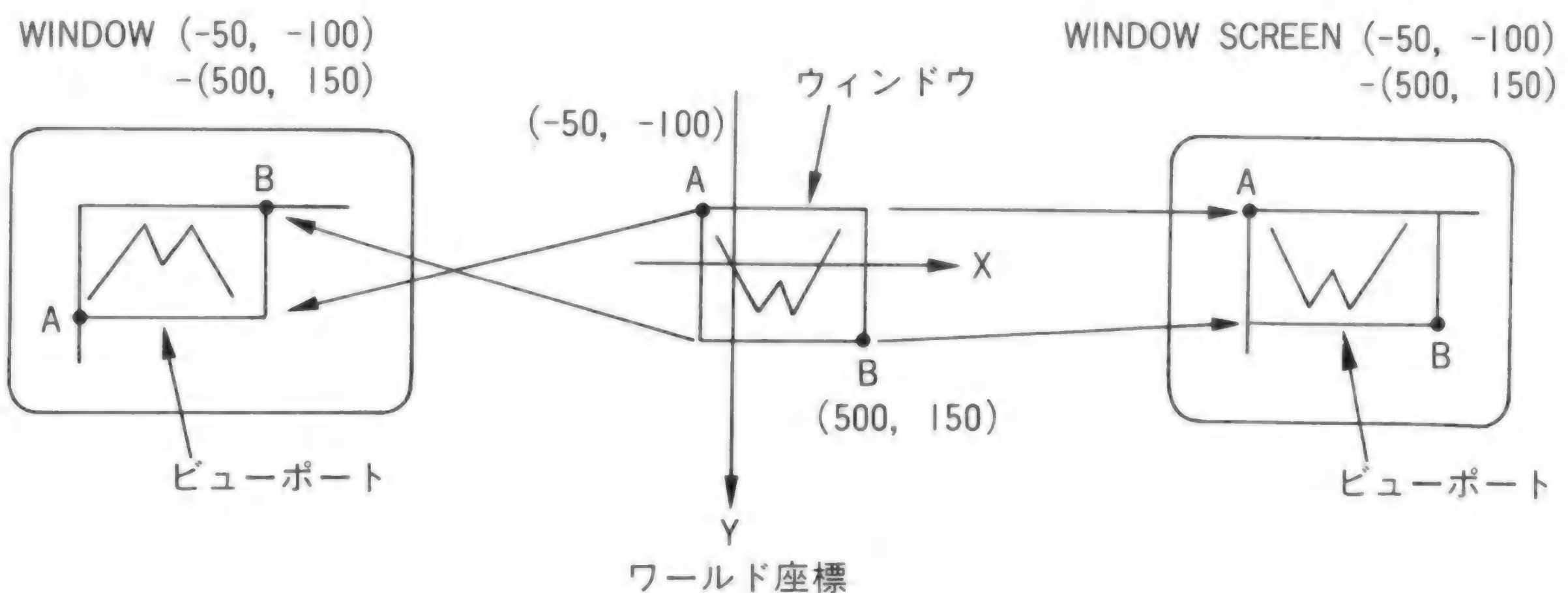
WINDOW [[SCREEN] (x1, y1) - (x2, y2)]

↑ウィンドウの座標

機 能

ワールド座標の(x1, y1) - (x2, y2)の範囲を、ウィンドウとして設定します。

Quick BASIC では、WINDOW 文によるウィンドウの切り方に 2 通りの方法があります。WINDOW 文に SCREEN オプションを付けると、下方を Y 座標の正の向きとするワールド座標をそのままの向きでビューポートに取り出し、SCREEN オプションを付けないと、Y 方向を逆転してビューポートに取り出します。



例

---/* ウィンドウの設定 */---

```
SCREEN 0: CLS
```

```
rd = 3.14159 / 180
```

```
x1 = -5: y1 = -5
```

```
x2 = 5: y2 = 5
```

```
VIEW (200, 100)-(400, 300), , 1 ← ビューポートの設定
```

```
FOR k = 1 TO 10
```

```
CLS
```

```
WINDOW (x1, y1)-(x2, y2) ← ウィンドウの設定
```

```
FOR j = 0 TO 360 STEP 4
```

```
  x = SIN(2 * j * rd): y = SIN(3 * j * rd)
```

```
  IF x = 0 THEN
```

```
    PSET (x, y)
```

```
  ELSE
```

```
    LINE -(x, y)
```

```
  END IF
```

```
NEXT j
```

```
x1 = x1 + .5: y1 = y1 + .5
```

```
x2 = x2 - .5: y2 = y2 - .5
```

```
NEXT k
```

← ウィンドウの範囲を狭めていく (図形は拡大されていく)

← リサージュ曲線の描画

**WRITE****(スクリーンへの出力)****書 式**

WRITE arg, ...
 ↑ 出力データ

機 能

arg で示される出力データをスクリーンに表示します。このとき、各データの区切りにカンマ(,) が挿入され、文字列の表示では、二重引用符(") で囲まれます。数値データを表示する際、前後に空白を付け加えません。

例

```
a = 10
b$ = "Hello"
c = 20
```

```
WRITE a, b$, c
PRINT a, b$, c
```

10,"Hello",20
10 Hello 20

WRITE # (シーケンシャル・ファイルへのライト)**書 式**

WRITE #n, arg, ...
 ↑ ↑ 出力データ
 ↑ ファイル番号

機 能

arg で示される出力データをファイル番号 n のファイルにライトします。各出力データはカンマ(,) で区切ります。出力に際して各データの区切りにカンマ(,) が挿入され、文字列データは、二重引用符(") で囲まれて出力されます。この点が PRINT #と異なります。

例

```
' ---/* シーケンシャルファイルの書き込み */---
```

```
OPEN "man.dat" FOR OUTPUT AS #1
```

```
INPUT "Name, year"; Name$, Year
```

```
DO WHILE Year >= 0
```

```
    WRITE #1, Name$, Year
```

```
    INPUT "Name, Year"; Name$, Year
```

```
LOOP
```

```
CLOSE #1
```

```
A>dump man.dat
```

```
Dump Version 3.00
```

00000000	22 41 6E 6E 22 2C 31 38-0D 0A 22 43 61 6E 64 79	"Ann",18.."Candy
00000010	22 2C 32 31 0D 0A 22 45-6C 75 7A 61 22 2C 31 39	",21.."Eluza",19
00000020	0D 0A 22 52 6F 6C 6C 61-22 2C 32 30 0D 0A	.."Rolla",20..

メタコマンド

\$INCLUDE

(ファイルのインクルード)

書 式

{REM | '}\$INCLUDE : 'filename'

↑ 取り込むファイル名

機 能

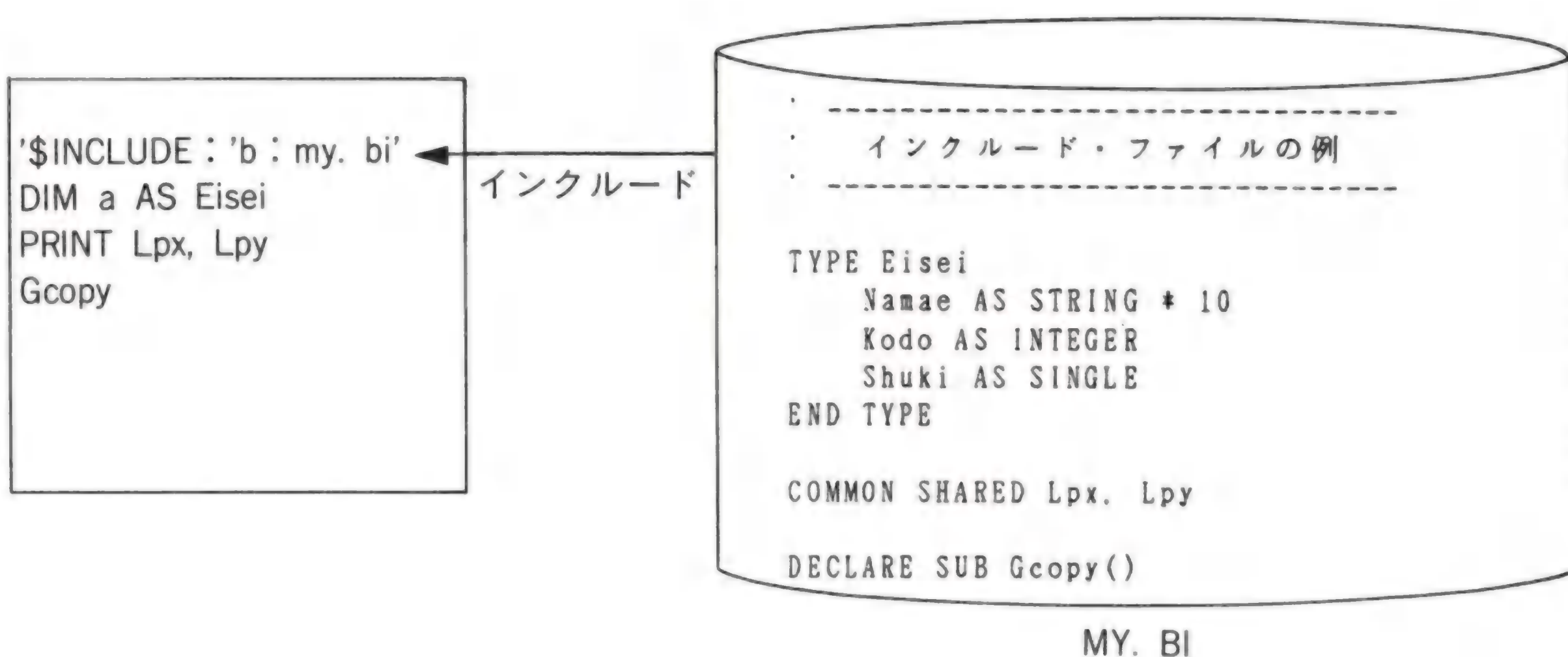
filename で示されるファイルを、この位置にインクルード(取り込み)します。

インクルード・ファイルは、テキストファイルでなければなりません。インクルード・ファイルの作り方は、第2章2-4の7を参照してください。プログラムをインクルード・ファイルにする場合は、保存形式を「T/テキストファイル」に指定してください。

インクルード・ファイルには、プロシージャ(SUB や FUNCTION)を記述できません。

filename のファイルタイプを省略すると、.BAS とみなされます。

例



\$DYNAMIC/\$STATIC

(動的配列／静的配列の指定)

書 式

- ①{REM |'}\$DYNAMIC
- ②{REM |'}\$STATIC

機 能

- ①以後の配列を動的配列として割り当てます。
- ②以後の配列を静的配列として割り当てます。

\$DYNAMIC/\$STATIC を用いて指定しない場合、定数の添字を用いて宣言した配列は静的配列、変数の添字を用いて宣言した配列または最初に COMMON ステートメントで宣言されている配列は動的配列となります。

例

```
'$DYNAMIC  
DIM a(100)      ← 動的配列
```

```
'$STATIC  
INPUT n  
DIM b(n)        ← 変数添字なので'$STATIC が指定されても動的配列  
DIM c(100)      ← 静的配列
```


第7章

Ver.4.2 から Ver.4.5 への 変更点

7-1

Ver.4.5における 主な変更点

Ver.4.2から Ver.4.5へのバージョンアップに伴い、変更された主な点は以下のとおりです。

- ・ PC-9801の最初期型(もっとも古いタイプ)は Ver.4.5ではサポートしない。
- ・ エミュレータモードで動作するインタプリタ QBE.EXE をサポート。
- ・ 強力なオンラインヘルプ(QB アドバイザ)のサポート。
- ・ PLAY, SOUND, UEVENT, SLEEP ステートメントの追加。
- ・ SCREEN, COLOR, WIDTH ステートメント, SCREEN, CSNG\$関数の機能変更。
- ・ 環境変数 QBGRPNEC の導入。
- ・ CIRCLE, LOCATE, PRINT ステートメントの高速化。
- ・ [STOP], [HELP] キーによるプログラム割り込みのサポート。
- ・ 統合環境のエディタのスクロールスピードの向上と、カーソル惰性の改善。
- ・ QB チュートリアルという学習ソフトのサポート。
- ・ ヘルプ作成ユーティリティ HELPMAKE のサポート。
- ・ ユーザライブラリとして汎用ライブラリ, グラフィックライブラリ, マウスライブラリをソースとライブラリでサポート。

7-2

Ver.4.5で提供されるソフトウェア

Ver.4.5では、セットアップディスク、プログラムディスク、ライブラリディスク、アドバイザディスクの4枚のフロッピーディスクが提供されます。以下のリストの中でコメントの付いているファイルが、Ver.4.5で追加された主なものです。

●セットアップディスク

ドライブ A: のディスクのボリュームラベルは QB45_SETUP
ディレクトリは A:Y

LEARN	<DIR>	89-11-15	4:50
SAMPLE	<DIR>	89-11-15	4:50
PACKING	LST	6974	89-11-15 4:50
QBE	DOC	4475	89-11-15 4:50
README	DOC	5532	89-11-15 4:50
SAMPLE	DOC	2942	89-11-15 4:50
SETUP	000	4099	89-11-15 4:50
SETUP	EXE	62193	89-11-15 4:50

8 個のファイルがあります。
614400 バイトが使用可能です。

ドライブ A: のディスクのボリュームラベルは QB45_SETUP
ディレクトリは A:YLEARN

.	<DIR>	89-11-15	4:50
..	<DIR>	89-11-15	4:50
BALL	BAS	1306	89-11-15 4:50
HELP	HLP	4302	89-11-15 4:50
LEARN	EXE	167100	89-11-15 4:50
MENU	ATB	4103	89-11-15 4:50
MENU	TXT	4103	89-11-15 4:50
PRINT	HLP	1510	89-11-15 4:50

← QB チュートリアル

8 個のファイルがあります。
614400 バイトが使用可能です。

ドライブ A: のディスクのボリュームラベルは QB45_SETUP
ディレクトリは A:YSAMPLE

.	<DIR>	89-11-15	4:50
..	<DIR>	89-11-15	4:50
ADVR_EX	<DIR>	89-11-15	4:50
BOOK	<DIR>	89-11-15	4:50
EXAMPLE	<DIR>	89-11-15	4:50
DEMO1	BAS	1934	89-11-15 4:50
DEMO2	BAS	2059	89-11-15 4:50
DEMO3	BAS	2140	89-11-15 4:50
QB	BI	1783	89-11-15 4:50
QCARDS	BAS	44644	89-11-15 4:50
QCARDS	DAT	2192	89-11-15 4:50
REMLINE	BAS	11712	89-11-15 4:50
SORTDEMO	BAS	20536	89-11-15 4:50
TORUS	BAS	24275	89-11-15 4:50
WALTZ	BAS	4254	89-11-15 4:50

15 個のファイルがあります。
614400 バイトが使用可能です。

●プログラムディスク

ドライブ A: のディスクのボリュームラベルは QB45_PGM
ディレクトリは A:Y

BIN <DIR> 89-11-15 4:50
MOUSE <DIR> 89-11-15 4:50

2 個のファイルがあります。
152576 バイトが使用可能です。

ドライブ A: のディスクのボリュームラベルは QB45_PGM
ディレクトリは A:YBIN

. <DIR> 89-11-15 4:50
.. <DIR> 89-11-15 4:50
BC EXE 111479 89-11-15 4:50
BRUN45A EXE 92329 89-11-15 4:50
LIB EXE 35643 88-07-26 10:52
LINK EXE 69133 88-09-07 16:27
QB EXE 311758 89-11-15 4:50
QB INI 48 89-11-15 4:50
EM <DIR> 89-11-10 4:50

← 27KB 大きくなった

9 個のファイルがあります。
152576 バイトが使用可能です。

ドライブ A: のディスクのボリュームラベルは QB45_PGM
ディレクトリは A:YBINYEM

. <DIR> 89-11-10 4:50
.. <DIR> 89-11-10 4:50
BRUN45E EXE 93113 89-11-15 4:50
QBE EXE 311870 89-11-15 4:50

← エミュレータ版インタプリタ

4 個のファイルがあります。
152576 バイトが使用可能です。

ドライブ A: のディスクのボリュームラベルは QB45_PGM
ディレクトリは A:YMOUSE

. <DIR> 89-11-15 4:50
.. <DIR> 89-11-15 4:50
MOUSE COM 8431 89-10-18 2:00
MOUSE DOC 47900 89-10-18 2:00
MOUSE SYS 8127 89-10-18 2:00

5 個のファイルがあります。
152576 バイトが使用可能です。

●ライブラリディスク

ドライブ A: のディスクのボリュームラベルは QB45_LIB
ディレクトリは A:Y

LIB <DIR> 89-11-15 4:50
USERLIB <DIR> 89-11-15 4:50
2 個のファイルがあります。
544768 バイトが使用可能です。

ドライブ A: のディスクのボリュームラベルは QB45_LIB
ディレクトリは A:YLIB

. <DIR> 89-11-15 4:50
.. <DIR> 89-11-15 4:50
EM <DIR> 89-11-15 4:50
ABSOLUTE ASM 4520 89-12-15 4:20
BCOM45A LIB 239343 89-11-15 4:50
BQLB45 LIB 25301 89-11-15 4:50
BRUN45A LIB 25769 89-11-15 4:50
INTRPT ASM 15099 89-12-15 4:20
QB LIB 2075 89-11-15 4:50
QB QLB 5814 89-11-15 4:50
10 個のファイルがあります。
544768 バイトが使用可能です。

ドライブ A: のディスクのボリュームラベルは QB45_LIB
ディレクトリは A:YLIBYEM

. <DIR> 89-11-15 4:50
.. <DIR> 89-11-15 4:50
BCOM45E LIB 229733 89-11-15 4:50
BRUN45E LIB 25769 89-11-15 4:50
4 個のファイルがあります。
544768 バイトが使用可能です。

ドライブ A: のディスクのボリュームラベルは QB45_LIB
ディレクトリは A:YUSERLIB

. <DIR> 89-11-15 4:50
.. <DIR> 89-11-15 4:50
ALL_MAKE BAT 544 89-11-15 4:50
DOSCALL OBJ 435 89-11-15 4:50
GEN BAS 16273 89-11-15 4:50
GEN BI 927 89-11-15 4:50
GEN LIB 13871 89-11-15 4:50
GEN QLB 13843 89-11-15 4:50
GENDEMO1 BAS 1094 89-11-15 4:50
GENDEMO2 BAS 4039 89-11-15 4:50
GEN_MAKE BAT 351 89-11-15 4:50
GRAPH BAS 6906 89-11-15 4:50
GRAPH BI 244 89-11-15 4:50
GRAPH LIB 6181 89-11-15 4:50
GRAPH QLB 9295 89-11-15 4:50
GRPDEMO1 BAS 592 89-11-15 4:50
GRPDEMO2 BAS 2518 89-11-15 4:50
GRP_MAKE BAT 261 89-11-15 4:50
MOUSE BAS 11059 89-11-15 4:50
MOUSE BI 576 89-11-15 4:50
MOUSE LIB 5675 89-11-15 4:50
MOUSE QLB 8083 89-11-15 4:50

汎用ライブラリ

グラフィックライブラリ

マウスライブラリ

MOUSEVNT ASM	2237	89-11-15	4:50
MOUSEVNT OBJ	225	89-11-15	4:50
MUSDEMO1 BAS	3392	89-11-15	4:50
MUSDEMO2 BAS	1385	89-11-15	4:50
MUS_MAKE BAT	281	89-11-15	4:50

27 個のファイルがあります。
544768 バイトが使用可能です。

●アドバイザディスク

ドライブ A: のディスクのボリュームラベルは QB45_ADVR
ディレクトリは A:¥

QBADVR <DIR> 89-11-15 4:50
1 個のファイルがあります。
549888 バイトが使用可能です。

ドライブ A: のディスクのボリュームラベルは QB45_ADVR
ディレクトリは A:¥QBADVR

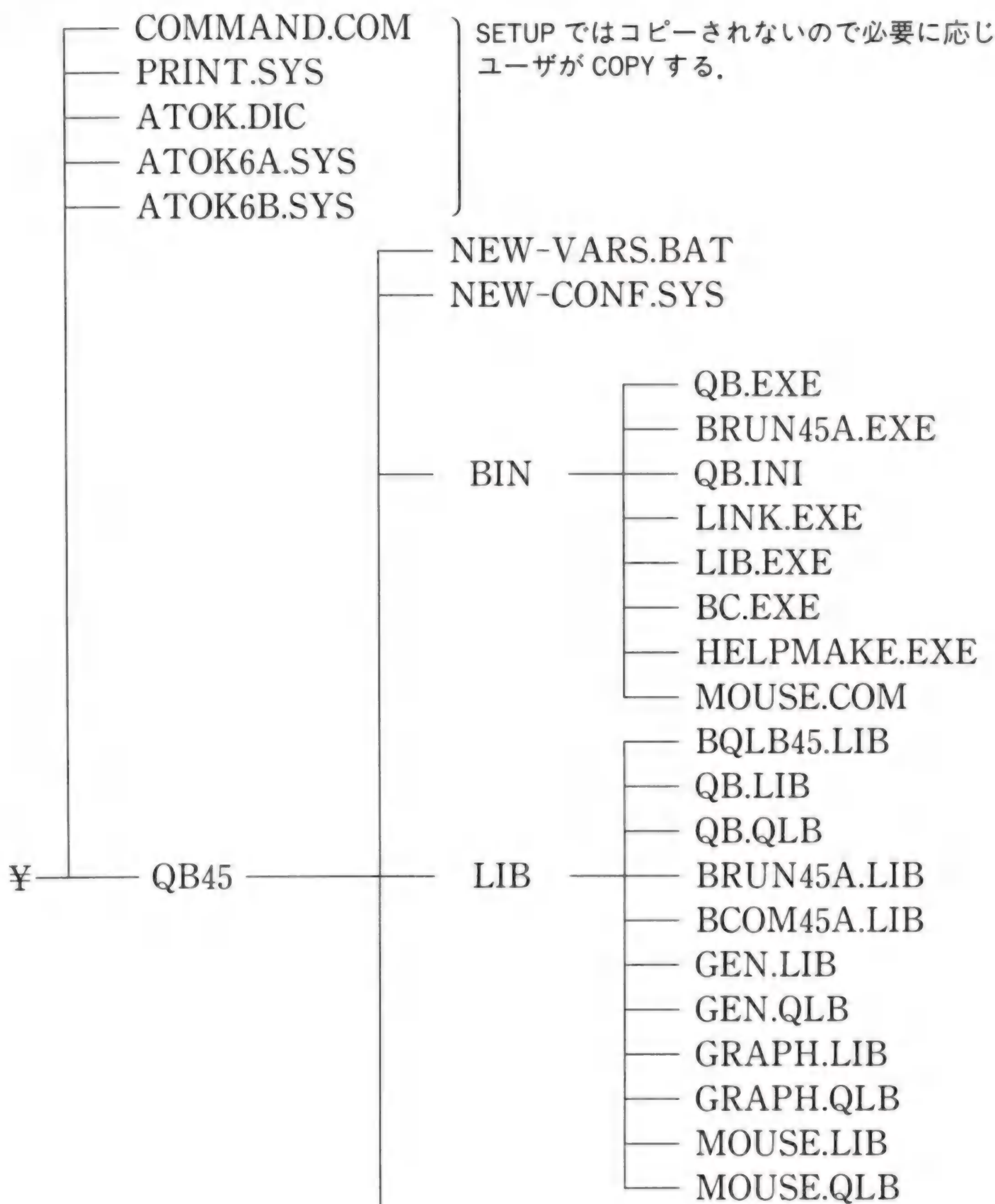
.	<DIR>	89-11-15	4:50	
..	<DIR>	89-11-15	4:50	
HELPMAKE EXE	47951	89-10-18	2:00	← ヘルプ作成ユーティリティ
QB45ADVR HLP	421739	89-11-15	4:50	← QB アドバイザ
QB45ENER HLP	69859	89-11-15	4:50	
QB45QCK HLP	121969	89-11-15	4:50	
QB45USER HLP	14216	89-11-15	4:50	
QB45USER QH	21450	89-11-15	4:50	

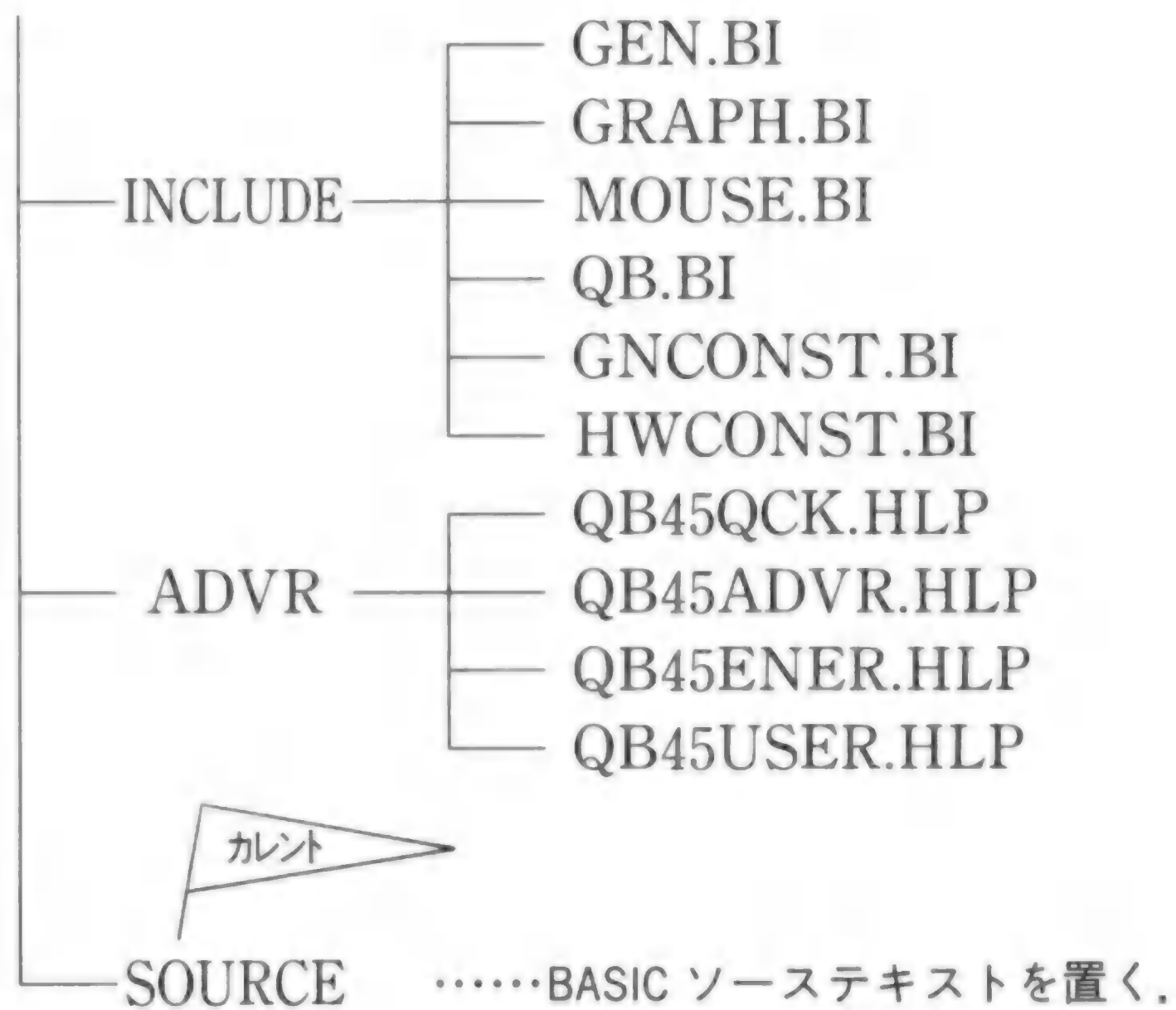
8 個のファイルがあります。
549888 バイトが使用可能です。

7-3 セットアップ

Ver.4.2と同様にSETUPユーティリティを用いて、フロッピーまたはハードディスクにQBシステムをセットアップできます。

■ハードディスク





SETUP により作成された NEW-VARS.BAT, NEW-CONF.SYS をもとに、上記構成を実現する AUTOEXEC.BAT と CONFIG.SYS を作ります。

NEW-VARS.BAT

```

A>type new-vars.bat
PATH=A:\YQB45\YBIN
SET LIB=A:\YQB45\YLIB
SET INCLUDE=A:\YQB45\YINCLUDE
SET TMP=A:\Y
MOUSE

```

NEW-CONF.SYS

```

A>type new-conf.sys
SHELL=A:\YCOMMAND.COM A:\Y /P
FILES= 20
BUFFERS=10

```

AUTOEXEC.BAT

```

A>TYPE YAUTOEXEC.BAT
PATH=A:\YQB45\YBIN
SET LIB=A:\YQB45\YLIB
SET INCLUDE=A:\YQB45\YINCLUDE
SET TMP=A:\Y
CD A:\YQB45\YSOURCE
MOUSE

```

CONFIG.SYS

```

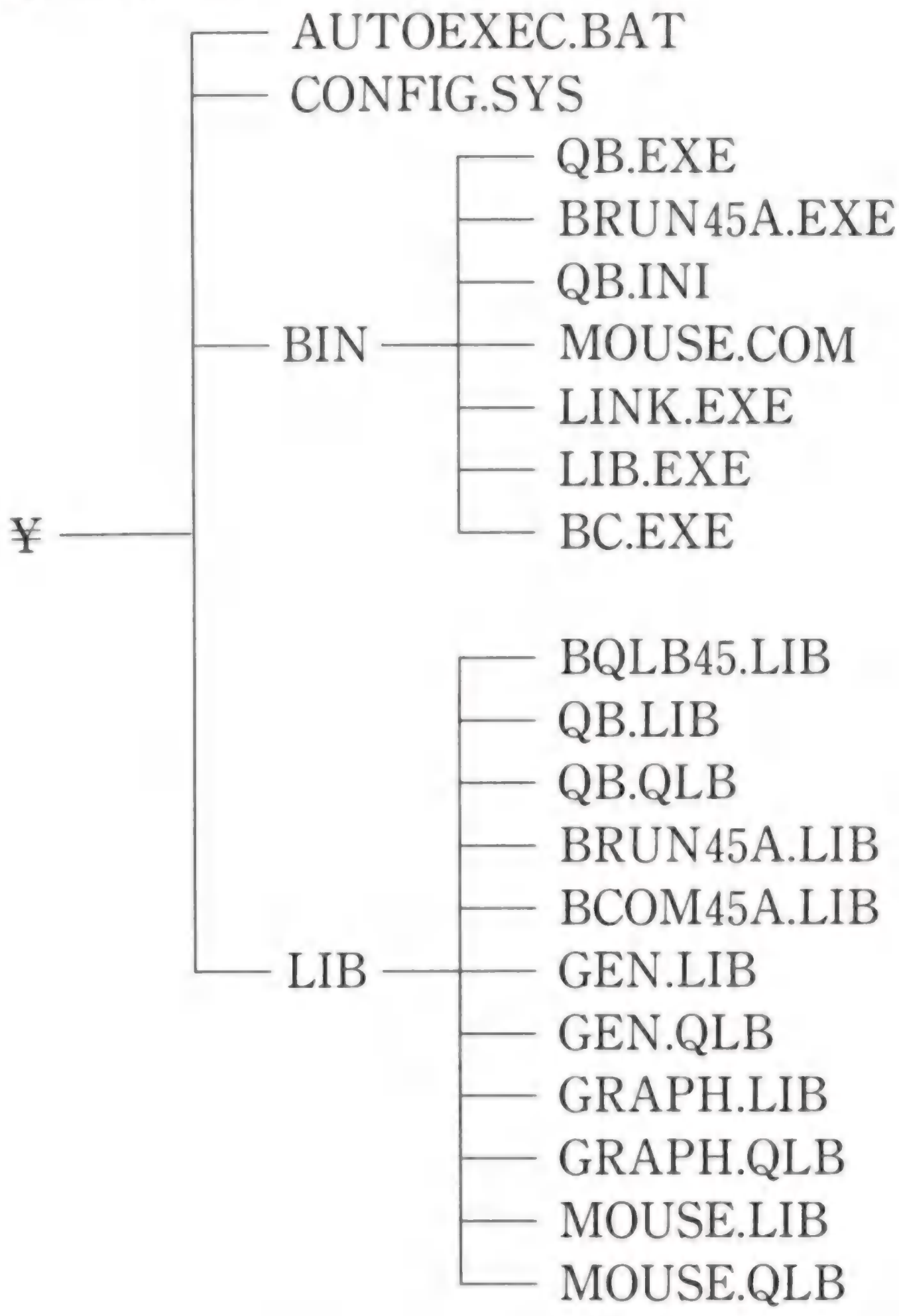
A>TYPE YCONFIG.SYS
SHELL=A:\YCOMMAND.COM A:\Y /P
FILES=20
BUFFERS = 10
DEVICE = ATOK6A.SYS
DEVICE = ATOK6B.SYS
DEVICE = PRINT.SYS

```

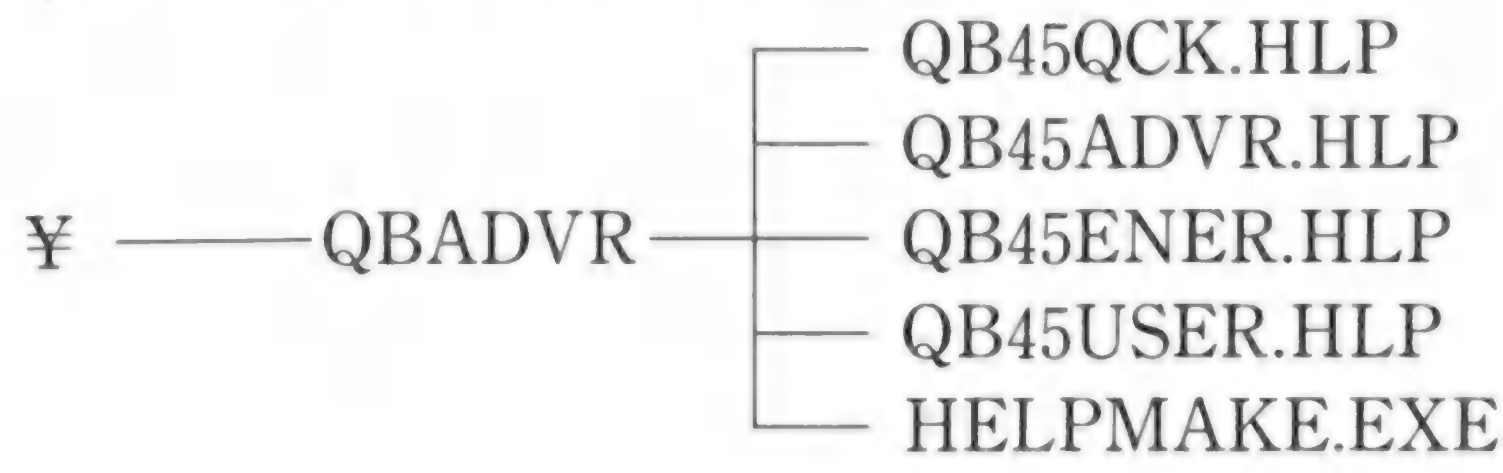

■ 1MB フロッピー (SETUP を用いた場合)

SETUP を用いてフロッピーにセットアップすると、次のような3枚のフロッピーに分けられてしまい使い勝手はよくありません。

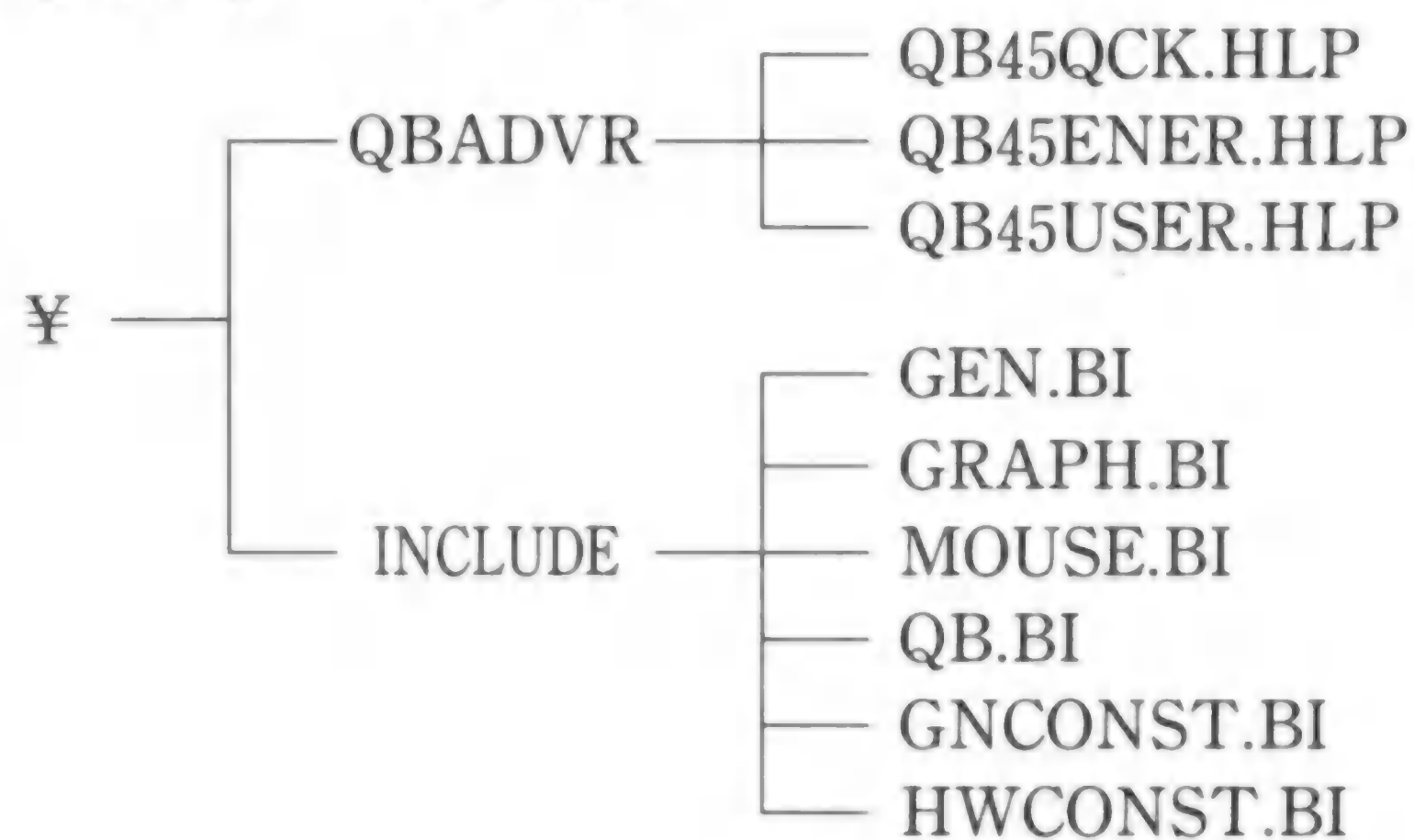
● QB 起動ディスク



● QB アドバイザディスク

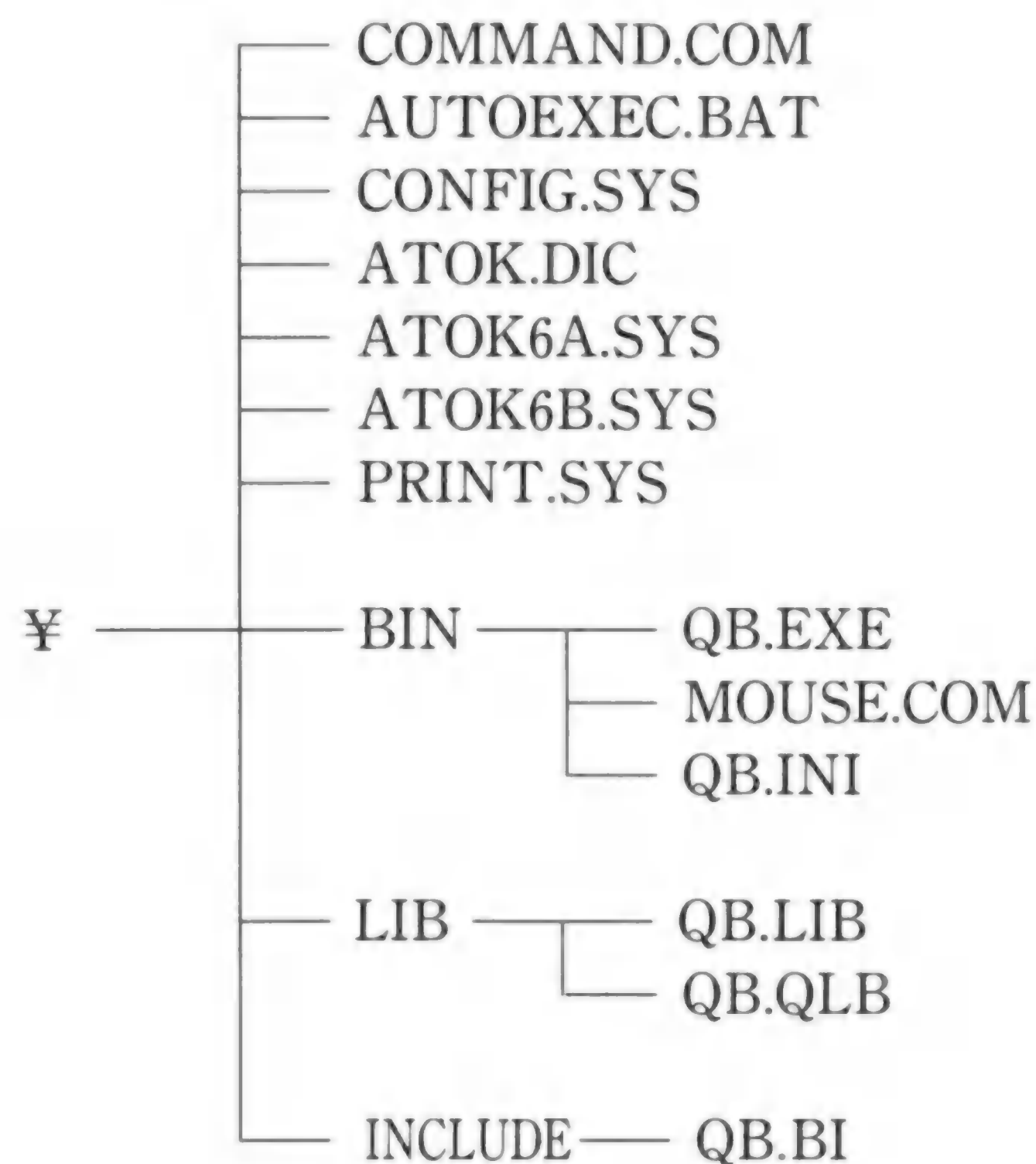


● QB ワークディスク



■ 1MB フロッピー (独自にコピーした場合)

次のように QB インタプリタだけのシステムにすれば、1枚のフロッピーに日本語辞書もいっしょに載せることができます。ただし、QB アドバイザは1枚に載せることはできません。



7-4 メニュー画面

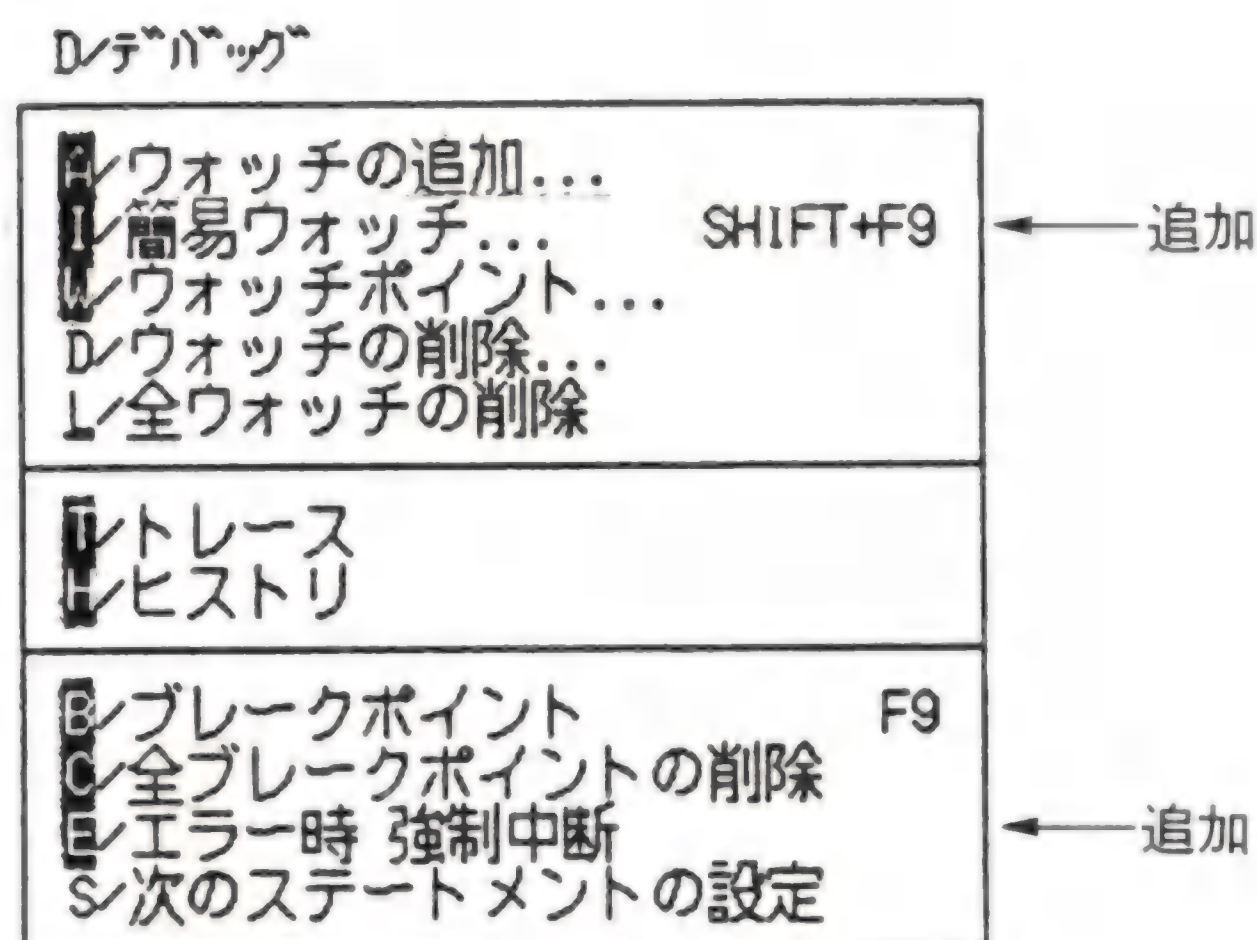
Ver.4.5では、QB 起動直後のメニュー画面は Easy メニューになっています。Easy メニューは、Full メニューのいくつかの機能を省略した初心者向けのメニューです。

Easy メニューと Full メニューの切り換えは、O/オプション — F/Full メニュー により行います。

Ver.4.2と Ver.4.5のメニュー画面は、基本的には変わりませんが、以下のメニューで相違があります。

■ D/デバッグ

I/簡易ウォッチ と E/エラー時強制中断 が追加されました。



● I/簡易ウォッチ

プログラムを途中で中断した際、そのときの変数の値や式の状態 (True なら 1, False なら 0 と表示) をダイアログボックスに表示します。ダイレクト・モードで変数の値を表示させるのと似た機能です。

式の値を表示します

式

sum ← 変数または式

値

100 ← 値または状態

● E/エラー時強制中断

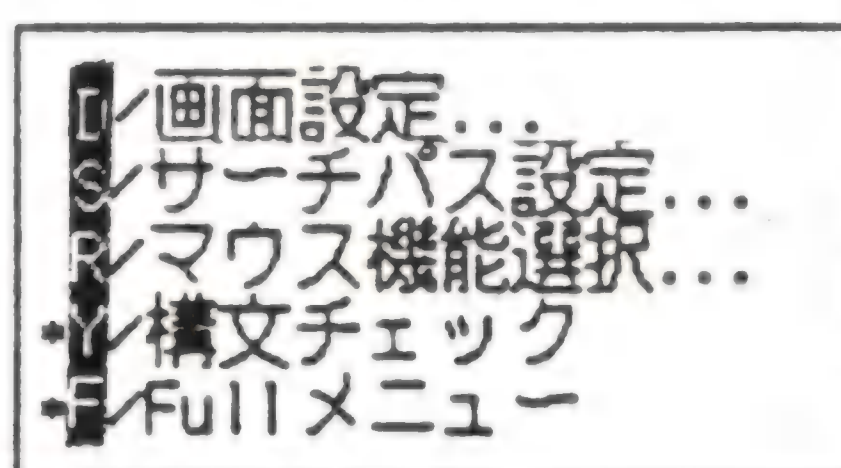
このサブメニューがONになっていると、ON ERROR 文で指定されているエラー・ハンドラの先頭にブレークポイントを設定します。

つまり、エラーが発生すると、エラー・ハンドラの先頭でブレークすることになります。

■ O/オプション

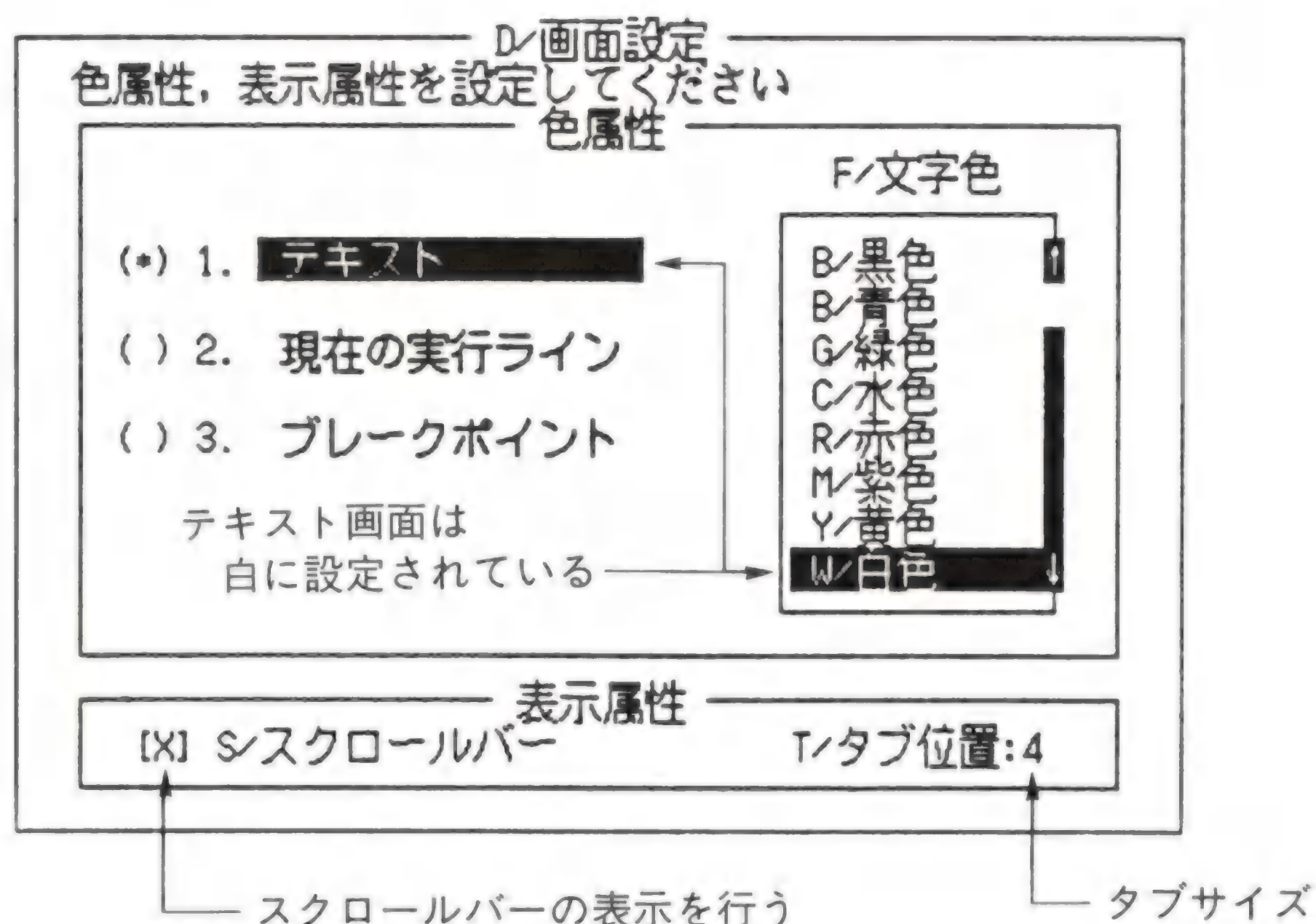
Ver.4.2では[V/表示]の中のサブメニューでしたが、Ver.4.5では独立したメニューとなり項目も追加されました。

O/オプション



● D/画面設定...

テキスト，現在の実行ライン，ブレークポイントの色，スクロールバーの表示／非表示，タブサイズについての設定を行います。



● S/サーチパス設定...

次のような各ファイルのサーチパスを設定します。

S/サーチパス設定

サーチパスを指定してください

X/実行ファイル:
(.EXE, .COM)

I/インクルードファイル:
(.BI, .BAS)

A:¥INCLUDE —パス名

L/ライブラリファイル:
(.LIB, .QLB)

E/ヘルプファイル:
(.HLP)

● R/マウス機能選択...

マウスの右ボタンをクリックしたときの機能を、次の2つのどちらかに設定します。

R/マウス機能選択

マウス右ボタン機能を設定してください

(*) C/キーワードヘルプの表示

() E/クリック行まで実行

【 確認 】

< 取消 >

< H/ヘルプ >

● Y/構文チェック

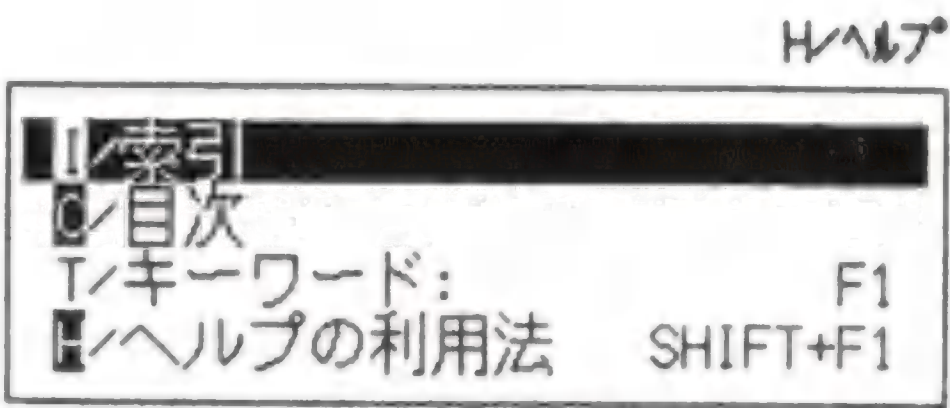
Ver.4.2では[E/編集]メニューに入っていましたが、Ver.4.5では[O/オプション]メニューに移されました。

● F/Full メニュー

Easy メニューと Full メニューの切り換えを行います。

H/ヘルプ (QB アドバイザ)

Ver.4.5では、QB アドバイザと呼ばれる強力なオンラインヘルプをサポートします。



I/索引

QB のキーワードをアルファベット順に分類し、選択したキーワードの説明を表示します。

HELP: 索引

《H/ヘルプの利用法》 《C/項目》 《I/索引》 《P/サポート》

QuickBASIC のキーワードの一覧を示します。キーワードのヘルプを参照するには、参照したいキーワードの先頭の1文字を押してから、[F・I]を押すと、キーワードの一覧が表示されます。参照したいキーワードへカーソルを移動してから再度、[F・I]キーを押して下さい。

A	B	C	D	E	F	G	H	I	J	K	L
M	N	O	P	R	S	T	U	V	W	X	

↓ A を選択し [F・I]

HELP: 索引-A

ABC	関数
ACCESS	キーワード
ALIAS	キーワード
AND	演算子
ANY	キーワード
APPEND	キーワード
AS	キーワード
ASC	関数
ATN Function	関数

↓ ABS を選択

HELP: ABS 関数:QuickSCREEN

《QuickSCREEN》 《D/詳細》 《E/プログラム例》 《C/目次》 《I/索引》

ABS - 数式の絶対値を返す数値演算関数です

構文

ABS(numeric-expression)

● C/目次

オンラインヘルプの内容の目次を表示します。

HELP: 目次
 《H/ヘルプの利用法》 《C/目次》 《I/索引》 《P/サポート》 《C/著作権》

《A/QuickBASIC の使い方》

- 《1. ショートカットキーの概要》
 - 《A/編集のためのキー操作》
 - 《B/画面表示のキー操作》
 - 《C/検索のキー操作》
 - 《D/実行とデバッグ操作》
 - 《E/ヘルプキー操作》
- 《2. QuickBASIC の制限事項》
- 《3. 以前のバージョンとの違い》
- 《4. QB 起動オプションスイッチ》
- 《5. Quickガイド》

《B/BASIC 言語仕様》

- 《1. 目的別キーワード一覧》
- 《2. 構文の表記の規則》
- 《3. プログラムの基本的な構成要素》
- 《4. データ型》
- 《5. 式と演算子》
- 《6. モジュールとプロシージャ》
- 《7. プログラミング例》
- 《8. 文字コード表》
- 《9. キースキャンコード》
- 《0. 拡張ライブラリの使い方》

● T/キーワード

ビューウィンドウのカーソル位置にあるキーワードのヘルプ情報を表示します。

● H/ヘルプの利用法

QB のオンラインヘルプの一般的な使い方を表示します。要約すると以下のとおりです。

- ・ヘルプを参照するには **[F・1]** を押すか、マウスの右ボタンを押す。
- ・ヘルプ情報から抜けるには **[ESC]** を押す。
- ・ヘルプ画面のスクロールは **[ROLL UP]**、**[ROLL DOWN]** を押す。
- ・《 》で囲まれた項目をハイパーリンクと呼び、この位置にカーソルを移し **[F・1]** を押すと、その項目のヘルプ情報が表示される。ハイパーリンク間の移動は **[TAB]** を押す。
- ・ヘルプ表示したハイパーリンクを20個まで記憶していて、**[GRPH] + [F・1]** で前に表示したハイパーリンクの内容を再び表示できる。

7-5

エミュレータ版
インタプリタ QBE.EXE

Ver.4.5では、エミュレータ・モードで動作するインタプリタ QBE.EXE をサポートしました。

	代替数値演算	エミュレータ
インタプリタ	QB.EXE	QBE.EXE
ランタイム・ライブラリ	BRUN45A.EXE BRUN45A.LIB	BRUN45E.EXE BRUN45E.LIB
スタンドアロン・ライブラリ	BCOM45A.LIB	BCOM45E.LIB

エミュレータ版において、マシンに数値演算コ・プロセッサがあると、次の演算処理が高速化されます。

- ・ 単精度/倍精度実数の四則演算
- ・ 3 角関数
- ・ 対数関数
- ・ 平方根

数値演算コ・プロセッサを搭載していないマシンで、QBE.EXE を使用した場合、数値演算の精度は、コ・プロセッサと同等の高い精度になりますが、演算スピードはQB.EXE より遅くなります。

コンパイルにおいて、代替数値演算とエミュレータ・モードを指定するには次のオプションで指定します。

/FPA … 代替数値演算
/FPI … エミュレータ

なお、QB.EXE からコンパイルすると自動的に/FPA、QBE.EXE からコンパイルすると自動的に/FPI が指定されます。

7-6 言語仕様上の変更点

1 追加されたステートメント/関数

Ver.4.5で追加されたステートメント関数を説明します。

PLAY

(音楽の演奏)

書 式

PLAY commandstring
 ↑ ミュージックコマンド文字列

機 能

commandstring で示される音符を演奏します。

コマンド	意 味
On	オクターブの設定. n=0~6
Nn	音符 n の演奏. n=0~84
<	1 オクターブ上げる
>	1 オクターブ下げる
A-G	現在のオクターブで A, B, ..., G を演奏. + : シャープ, - : フラット
Ln	音符の長さを n 分音符に設定. n=1~64
Tn	1 分間当りの四分音符の数を設定. n=32~255
MS	各音符を3/4の長さで演奏
MN	各音符を7/8の長さで演奏
ML	各音符を完全な長さで演奏
Pn	n 個の四分音符の休止. n=1~64
MF	フォアグラウンドで音楽を演奏
MB	バックグラウンドで音楽を演奏
X	X+VARPTR\$(str) で str で示される文字列変数のデータをコマンドとして演奏する

例

PLAY (音符の残数の取得)

関数

書式

PLAY(n)

「タミー」

機能

演奏バッファ(キュー)に残っている音符の数を返します。

例

```
DO WHILE PLAY(0)< >0
LOOP
```

…演奏バッファの音符が残らず演奏されるまで待つ。

ON PLAY (音符残数によるトラッピング)

書式

ON PLAY(limit) GOSUB line

分岐限界の音符数(1~32)

機能

BGM キュー(バッファ)の残り音符数が limit を切ったとき, line で示されるイベント・トラッピングルーチンへ分岐します.

PLAY ON/OFF/STOP(音符残数トラッピングの制御)

書 式

- ① PLAY ON
- ② PLAY OFF
- ③ PLAY STOP

機 能

- ① ON PLAY によるイベント・トラッピングを可能にします.
- ② ON PLAY によるイベント・トラッピングを中止します.
- ③ ON PLAY によるイベント・トラッピングを中断します.

SOUND

(音の発生)

書 式 SOUND freq, time

↑ ↑

音の発生期間

音の周波数(Hz) .37~32767

機 能

周波数 freq の音を time で示される期間だけ発生します. time の 1 単位は約0.055秒です. 1 秒を指定するには time に18.2を指定します.

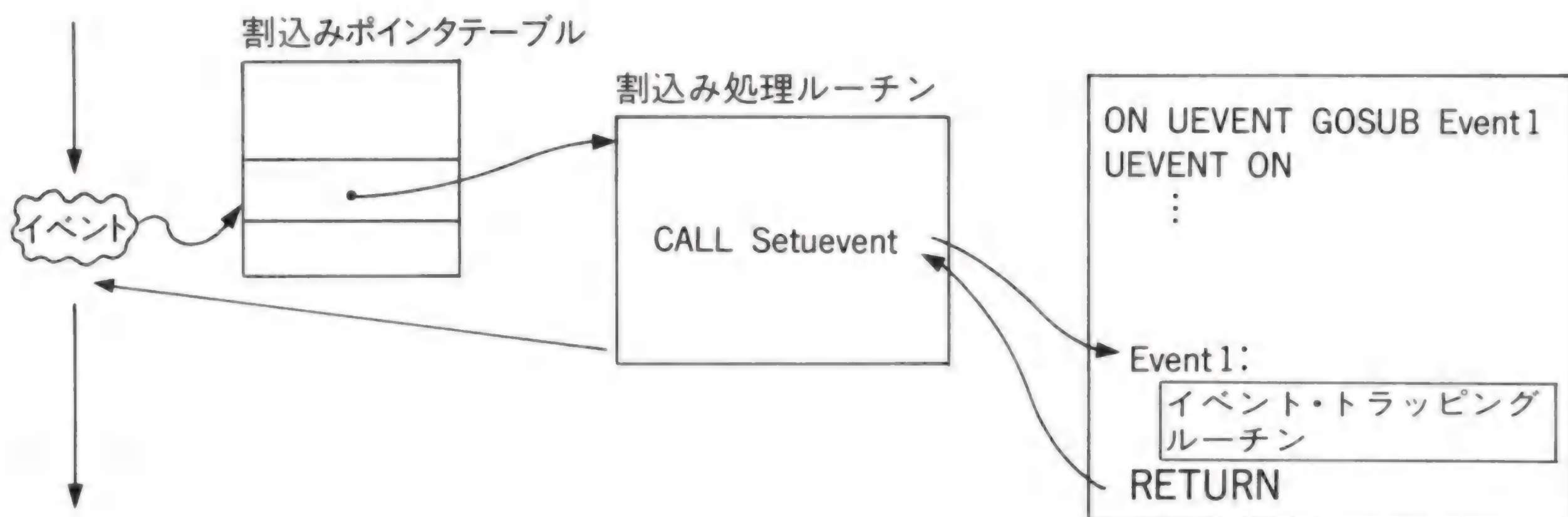
注) PC-9801E/F/M/U では音程を変えることはできません.

ON UEVENT (ユーザ定義イベントによるトラッピング)

書 式 ON UEVENT GOSUB line

↑ 分岐する行番号またはラベル

機 能



ユーザ定義イベントが起こると、割り込みポインタテーブルに設定されているエントリ・アドレスにしたがって割り込み処理ルーチンへ分岐します。その中でCALL Setueventを行うと、ON UEVENTで設定したイベント・トラッピングルーチンへ分岐します。

割り込みポインタテーブルのエントリ・アドレスの設定と、割り込み処理ルーチンはユーザが行わなければなりません。

Setuevent プロシージャは、QB.QLB の中に入っています。

UEVENT ON/OFF/STOP (ユーザイベント・トラッピングの制御)

書 式

- ① UEVENT ON
- ② UEVENT OFF
- ③ UEVENT STOP

機 能

- ①ユーザイベント・トラッピングを可能にします。
- ②ユーザイベント・トラッピングを中止します。
- ③ユーザイベント・トラッピングを中断します。

SLEEP (スリープ)

書 式 SLEEP sec
 ↑ スリープする秒

機 能

sec 秒だけプログラムの実行を中断(スリープ)します。ただし、スリープ中に何かキーが押されるか、プログラムの実行を再開するイベントが起こると、スリープは解除されます。

sec を省略すると、何かキーが押されるか、プログラムの実行を再開するイベントが起こるまでスリープします。

2 変更されたステートメント/関数

Ver.4.5において、機能が変更されたステートメントおよび関数について説明します。

SCREEN (画面モードの設定)

書 式 SCREEN mode, paletteswitch, apage, vpage

機 能 mode に82を追加しました。
paletteswitch に 8 色中 8 色のパレットモードを指定する 0 を追加しました。
mode, paletteswitch, apage, vpage に指定できるパラメータは、次のようなマシン群により異なります。

- ① PC-9801 E/F/M/VM2/VX2/UV2/VM21 (VM11,UV21は含まない)
- ② PC-9801 U
- ③ 上記以外

mode	paletteswitch			apage			vpage		
	①	②	③	①	②	③	①	②	③
81	×	×	×	×	×	×	×	×	×
82	×	×	×	×	×	×	×	×	×
84	0	0	0-3	0-3	0-1	0-3	0-1	0-1	0-3
87	0	0	0-3	0-1	0	0-1	0-1	0	0-1
88, 0	0	0	0-3	0-1	0	0-1	0-1	0	0-1

COLOR (表示色の指定)

書 式 COLOR fore, back

機 能 スーパーインポーズ・モード (mode = 82, 84, 88, 0) のときの、COLOR 文の第 2 パラメータ back の機能が次のように変更されました。
fore = 0 のときだけ back を次のように指定できます。

0	1	2	3	4	5	6	7
黒	青	緑	水	赤	紫	黄	白

WIDTH

(画面幅の設定)

書 式 WIDTH column, row

機 能 Ver.4.2では row として25だけでしたが, Ver.4.5では20と25をサポートします. デフォルトは25行モード.

SCREEN

(指定位置の文字または色の取得) **関数**

書 式 SCREEN(row, column[, colorflag])

機 能 Ver.4.2では, SCREEN で文字コードを得る場合, 漢字の1バイト目と2バイト目を区別できませんでしたが, Ver.4.5では区別できるようになりました.

つまり, 指定位置が1バイト目のときは, その文字の漢字連続コードを, 2バイト目のときは, 負の漢字連続コードを返します.

CSNG\$

(2バイト文字を1バイト文字に変換) **関数**

書 式 CSNG\$(str)

機 能 Ver.4.5では, 全角カタカナの濁音, 破裂音, 促音の半角への変換と, 全角ひらがなの半角への変換をサポートしました.

バ	→	ハ
ギャ	→	キャ
あ	→	ア

3 環境変数 QBGRPNEC の導入

A>SET QBGRPNEC = YES

などと、環境変数 QBGRPNEC を定義しておくと、パレットの順番を PC-9801 のオリジナルパレットとコンパチブルにします。

ただし、テキスト画面の色は PC-9801 とコンパチとはなりません。

0	1	2	3	4	5	6	7
黒	青	赤	紫	緑	黄	水	白

また、QBGRPNEC が定義されていると、次のような動作をとります。

- ・プログラム実行/終了時にグラフィック画面のクリアを行わない。
- ・プログラム実行/終了時にパレットを初期化しない。

4 STOP / HELP キーによるプログラム割り込みのサポート

STOP キーでの割り込みを、

ON KEY(30) GOSUB line

HELP キーでの割り込みを、

ON KEY(31) GOSUB line

で行えるようにしました。

5 ソーステキスト、ライブラリの互換性

標準ファイルでセーブしたソーステキストは、Ver.4.2 のものを Ver.4.5 でも読めますが、Ver.4.5 のものを Ver.4.2 で読むことはできません。テキスト・ファイルの形式で保存してあるものはどちらでも読めます。

Ver.4.2 で作成したライブラリを、Ver.4.5 で利用できませんし、その逆もできません。利用するバージョンに合わせて、ライブラリを再構築してください。

7-7

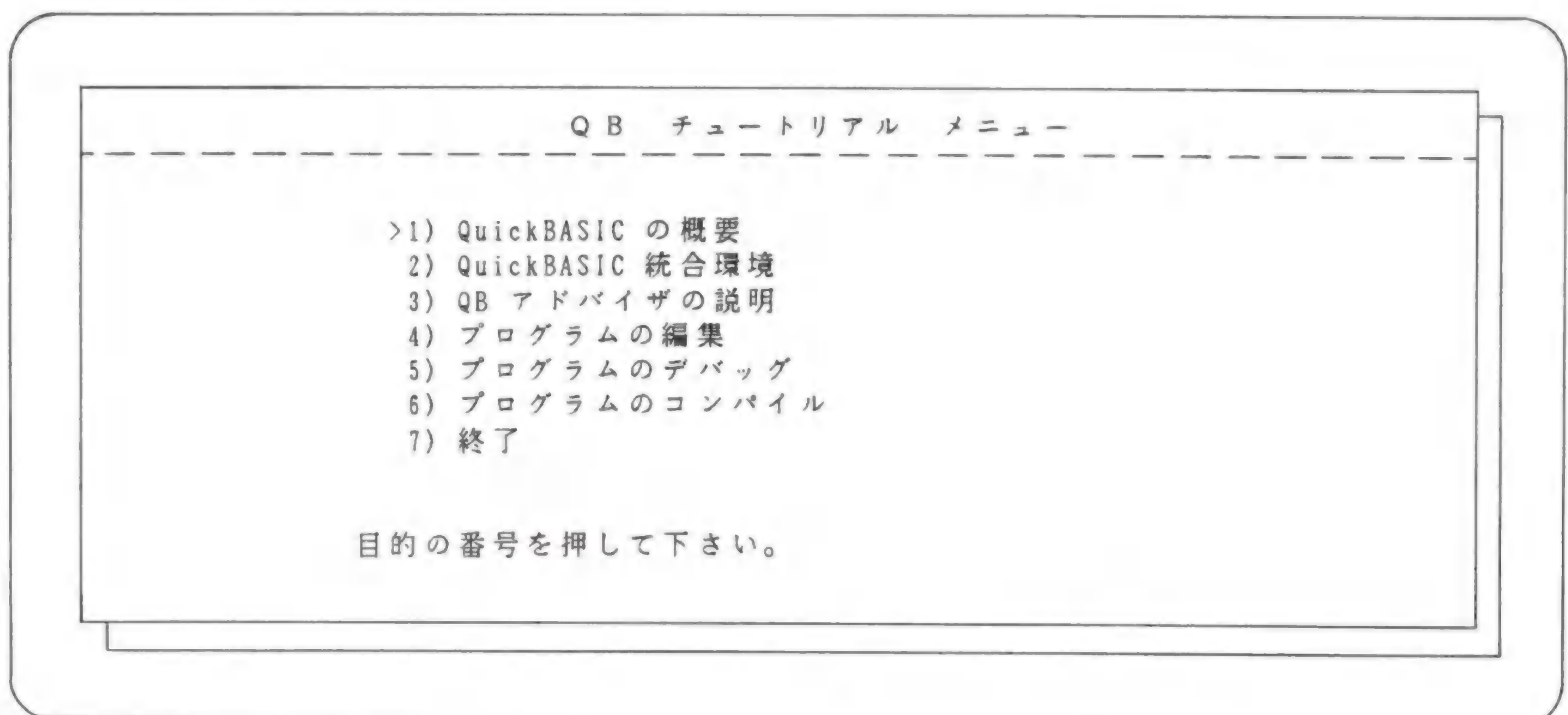
QB チュートリアル

Ver.4.5では、QB の概要を画面上で学習させるソフトとしてQB チュートリアルを提供しました。

セットアップディスクをカレントドライブに置き、サブディレクトリ LEARN をカレントディレクトリにして、

A>LEARN

とすることで、次のようなQB チュートリアルを起動できます。



なお、SETUP ユーティリティを起動し、そこからQB チュートリアルに入ること
もできます。

7-8 HELPMAKE

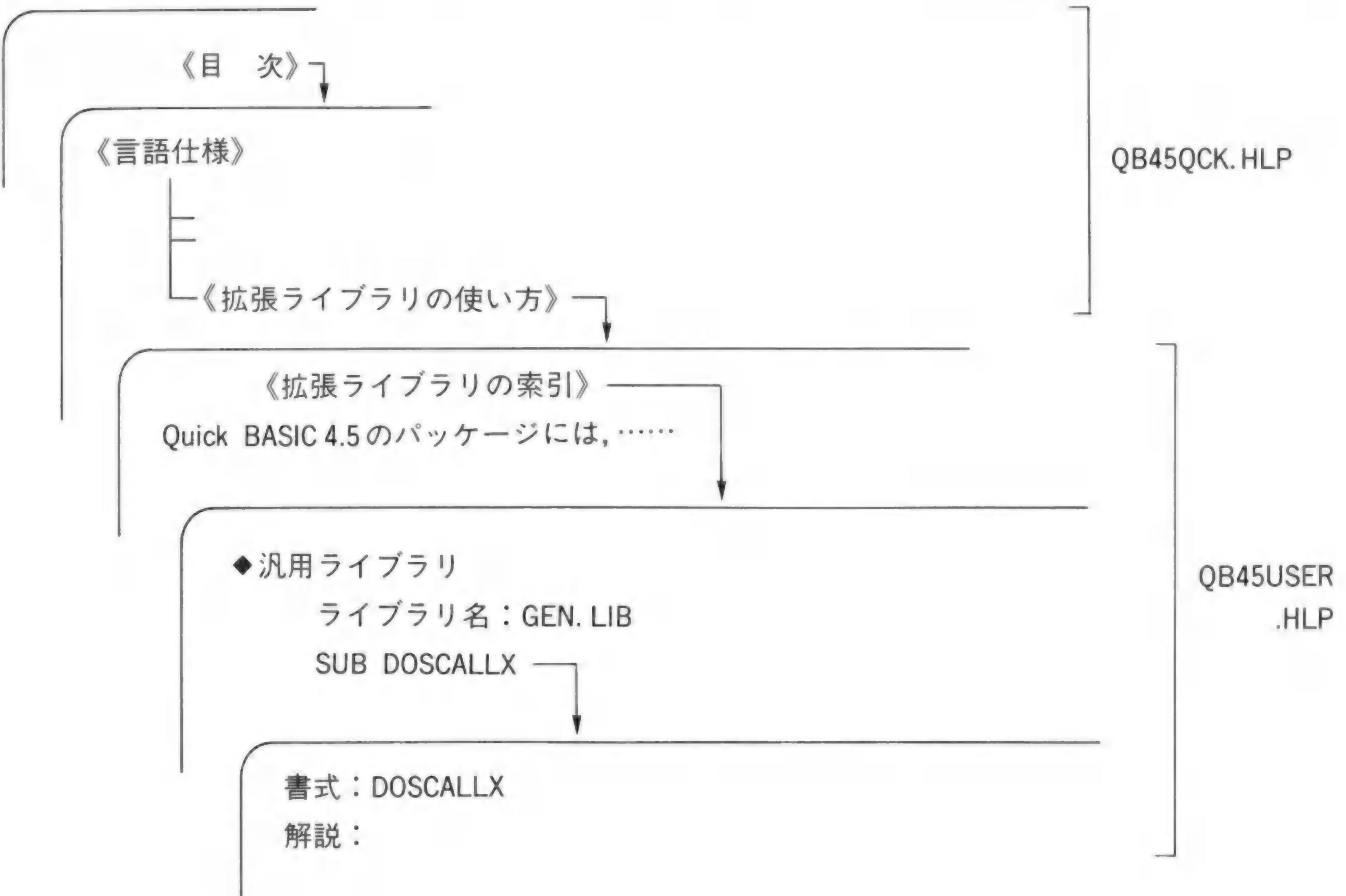
1 ヘルプファイルの概要

HELMAKE は、ヘルプファイルを作成するためのユーティリティです。
QB が標準で使うヘルプファイルは次の 4 つです。

- QB45QCK.HLP ... 索引、目次、関数、ステートメントのトピックが記述されている。
- QB45ENER.HLP ... メニューやダイアログボックスの解説、エラーの解説
- QB45ADVR.HLP ... 関数、ステートメントの詳しい解説やプログラム例
- QB45USER.HLP ... ユーザライブラリの説明

さて、このヘルプファイルは、ファイルサイズを小さくするために圧縮(パック)されていますので、エディタで直接編集することはできません。ヘルプファイルの内容を変更したい場合は、HELMAKE を用いて一度テキスト・ファイルに変換し、必要な修正を行ったあと、再び HELPMAKE を用いて、逆にテキスト・ファイルをヘルプファイルに変換します。

次の図は、拡張ライブラリの DOSCALLX のヘルプを見る過程を示したものです。



《 》の中と SUB DOSCALLX (緑色で表示される) はハイパーリンクになっています。

さて、これらのヘルプ参照はハイパーリンクにより結合されているわけですが、その構造を以下に示します。

QB45QCK.HLP をテキスト・ファイルに変換してスクリーンに表示してみましょう。

```
A>helpmake /d b:qb45qck.hlp
Microsoft (R) Help File Maintenance Utility
Version 1.00
Copyright (c) Microsoft Corp 1988-1989. All rights reserved.
Decoding: b:qb45qck.hlp
.
.
.
Yi 《YaYpB/BASIC 言語仕様Yi》 Yvqb45advr.hlp!.kemenyYvYp
├── Yi 《YaYp1. 目的別キーワード一覧Yi》 Yvqb45advr.hlp!.fkYvYp
├── Yi 《YaYp2. 構文の表記の規則Yi》 Yvqb45advr.hlp!.tokYvYp
├── Yi 《YaYp3. プログラムの基本的な構成要素Yi》 Yvqb45advr.hlp!.funYvYp
├── Yi 《YaYp4. データ型Yi》 Yvqb45advr.hlp!.dtpYvYp
├── Yi 《YaYp5. 式と演算子Yi》 Yvqb45advr.hlp!.exoYvYp
├── Yi 《YaYp6. モジュールとプロシージャYi》 Yvqb45advr.hlp!.mxpYvYp
├── Yi 《YaYp7. プログラミング例Yi》 Yvqb45advr.hlp!.pswitchYvYp
├── Yi 《YaYp8. 文字コード表Yi》 Yvqb45advr.hlp!.acYvYp
├── Yi 《YaYp9. キースキャンコードYi》 Yvqb45advr.hlp!.kbsetYvYp
└── Yi 《YaYp0. 拡張ライブラリの使い方Yi》 Yvqb45user.hlp!.userlibhYvYp
                                         ↑      ↑
                                         ヘルプファイル名   ハイパーリンク ID
```

QB45USER.HLP のテキスト・ファイルは、QB45USER.QH として提供されていますので、TYPE で表示してみましょう。


```
A>type qb45user.qh
.context .userlibh ← ハイパーリンク ID
.context @userlibh
:n 拡張ライブラリの概要
Yi <YiYaC/目次Yi> YiYv-9996YvYp Yi <YiYaI索引Yi> YiYv-9997YvYp Yi <YiYaU/拡張
ライブラリの索引Yi> Yiv@userlibcYvYp
      ↙      ↘
      ハイパーリンク      ハイパーリンク ID

.context .userlibc
.context @userlibc ← ハイパーリンク ID
:n 拡張ライブラリの索引

◆ 汎用ライブラリ
   ライブラリ名          : GEN.LIB
   クイックライブラリ名 : GEN.QLB
   インクルードファイル : QB.BI, GEN.BI

   YiYaSUB      DOSCALLX YiYv@DOSCALLYvYp      INT 21Hを呼び出しま
す. DS/ES対応 ↙      ↘      ハイパーリンク ID
   YiYaSUB      DOSCALL YiYv@DOSCALLYvYp      INT 21Hを呼び出しま
す.

.context @DOSCALL ← ハイパーリンク ID
.context @DOSCALLX
:n DOSCALL/DOSCALLX SUB PROGRAM
DOSCALL/DOSCALLX の利用法

ライブラリ名          : GEN.LIB
クイックライブラリ名 : GEN.QLB
インクルードファイル : QB.BI, GEN.BI

構文: DOSCALL[X] inreg, outreg
```

ヘルプテキストの構成の概要を以下に示します.

```
.context   ハイパーリンク ID

:n 見出し

   説明文
   :
   ¥a   ハイパーリンク   ¥v   ハイパーリンクID
```

各説明文は、.context で始まり、後の<ハイパーリンク ID>がタイトルの役割をし、他の説明文中のハイパーリンクからの参照を可能にします。:n の後の<見出し>はヘルプ画面に表示されたときの見出しになります。

説明文中にハイパーリンクを設定するには、ハイパーリンクにする項目を¥a と¥v で囲み、そのあとにそのハイパーリンクへの<ハイパーリンク ID>を書きます。

2 ヘルプファイルの変更例

ユーザライブラリヘルプ(QB45USER.HLP)に新しい説明文を追加する具体例を以下に示します。

● QB45USER.QH(テキスト・ファイル)

◆ グラフィックライブラリ

ライブラリ名 : GRAPH.LIB
クイックライブラリ名: GRAPH.QLB
インクルードファイル: QB.BI, GRAPH.BI

YiYaSUB GROLLYiYv@GROLLYvYp
YiYaSUB KPUTYiYv@KPUTYvYp
YiYaSUB LINEXORYiYv@LINEXORYvYp

グラフィック画面をス
漢字を描画します。
XOR モードで線を引

* プリンタライブラリ

ライブラリ名 : PRINTER.QLB

YiYaSUB LPUTCYiYv@LPUTCYvYp
:
:
:

プリンタへの1文字出力 } ←追加した
説明文

ハイパーリンク ID

.context @LPUTC ←ハイパーリンク ID
:nPRINTER SUB PROGRAM

書式 : LPUTC c%

解説 : c%で示される文字をプリンタに出力します。 } ←追加した
説明文

修正したヘルプテキスト・ファイル(QB45USER.QH)を、ヘルプファイル(QB45USER.HLP)に変換する様子を以下に示します。

制御文字 (必ず/a:と指定)

ファイル圧縮率を最大 (約50%)

出力ファイル

ヘルプファイルの1行の最大長

テキスト・ファイル

```
A>helpmake /a: /e /ob:qb45user.hlp /w150 qb45user.qh
Microsoft (R) Help File Maintenance Utility
Version 1.00
Copyright (c) Microsoft Corp 1988-1989. All rights reserved.
```


3 HELPMAKE の仕様概要

● HELPMAKE の起動

A>HELPMAKE [[option] {/En | /D} [option]] sourcefile

● オプション

		意 味
エンコードオプション	/Ac	ヘルプデータベースの制御文字として c を指定. QB ではつねに /A: と指定 ファイル圧縮率. n は 0~15. n を省略すると最高の圧縮率 (約 50%)
	/En	
	/H	
	/L	
	/Odestfile	作成するヘルプファイルがデコード (アンパック) されないようにロックする
	/S2	出力するヘルプファイル名
	/Vn	QB 用オプションで省略できる
デコードオプション	/Wn	HELPMAKE の処理過程の表示方法のレベルを n (0~6) で指定. n を省略すると全過程の表示
	/D	ヘルプファイルのデコード
	/DU	ヘルプファイルのデコード デコードしたテキストファイル中のクロスリファレンス, 表示指定情報を削除
	/H	HELPMAKE のヘルプ
	/Odestfile	作成するテキスト・ファイル名. 省略するとスクリーンに出力
	/Vn	HELPMAKE の処理過程の表示方法のレベルを n (0~3) で指定. n を省略すると全過程の表示

● 特別な context 名

context	説 明
h.default	デフォルトのヘルプ画面
h.notfound	要求された context が見つからないときに表示するトピック
h.pg1	論理的に最初にあるトピック
h.pg\$	論理的に最後にあるトピック
h.	ヘルプ項の参照先であることを示す
m.	メニュー項目のヘルプの参照先であることを示す
e.	エラーのヘルプ参照先であることを示す

●テキストファイル中の制御文字

制 御 文 字	意 味
:ln	トピックを表示するウィンドウの行数を n で指定
:n text	ヘルプのタイトルを text で指定
¥a	ハイパーリンクの先頭
¥b, ¥B	太字指定のオン/オフ
¥i, ¥I	イタリック指定のオン/オフ
¥p, ¥P	すべての属性をオフ
¥u, ¥U	下線指定のオン/オフ
¥v, ¥V	ハイパーリンクの終わり
¥¥	¥マーク自身

7-9 ユーザライブラリ

Ver.4.5では次のようなユーザライブラリが提供されています。

GEN.QLB(GEN.LIB)	…汎用ライブラリ
GRAPH.QLB(GRAPH.LIB)	…グラフィックライブラリ
MOUSE.QLB(MOUSE.LIB)	…マウスライブラリ

これらのライブラリは、～.QLB (クイックライブラリ)、～.LIB (ライブラリ)、～.BI(プロシージャ宣言ファイル)、～.BAS (ソーステキスト)の形態で提供されています。使用にあたってはライブラリをLIBディレクトリ、インクルードファイルをINCLUDEディレクトリに置き、たとえばGRAPH.QLBを使用するには、

A>QB /LGRAPH

と起動します。 ユーザプログラムでは、 QB.BI, GRAPH.BI をインクルードします。

```
' $INCLUDE: 'qb.bi'  
' $INCLUDE: 'graph.bi'  
  
SCREEN 0: CLS  
KPUT 100, 100, "日本語のグラフィック画面への表示", 1, 0  
LINE (90, 95)-(370, 120), 5, B
```

日本語のグラフィック画面への表示

各ライブラリの仕様ドキュメントは、ソーステキスト(GEN. BAS, GRAPH. BAS, MOUSE. BAS)に記載されています。

以下にその仕様の要約を説明します。

1 汎用ライブラリ

GEN.QLB のライブラリについて説明します。このライブラリを使用するユーザプログラムでは、QB.BI と GEN.BI をインクルードしてください。

Fep

(FEP の ON/OFF)

書 式 Sep f\$
 ↑ "ON" または "OFF"

機能 FEP を ON または OFF します。

GetCommand\$

(コマンドライン引数の取得)

関数

書式 GetCommand\$(n%)
 ↑ 引数の番号

機 能 n%番目のコマンドライン引数を取得します。

GetFile\$

(ディレクトリ・エントリの検索)

関数

書式

```
GetFile$ (FileName$, Attr%, DtaData AS DtaType, Func%)
```

- ↑ 0, 非0
- ↑ ディレクトリ・エントリ用バッファ
- ↑ ファイル属性
- ↑ 検索するファイル名。ワイルドカードを含む

機能	<p>FileName\$で示されるファイルのディレクトリ・エントリを検索し、DtaData に返します。</p> <p>GetFile\$の戻り値は取得されたファイル名です。</p>	
----	--	--


```
TYPE DtaType
  DtaDummy AS STRING * 21
  DtaAttr AS STRING * 1
  DtaTime AS STRING * 2
  DtaDate AS STRING * 2
  DtaSize AS LONG
  DtaName AS STRING * 14
END TYPE
```

Attr%には検索されたファイルの属性が次のような値で返されます。

- 0 通常ファイル
- 1 リードオンリー・ファイル
- 2 隠しファイル
- 4 システム属性
- 8 ボリューム名
- &H10 ディレクトリ
- &H20 アーカイブ

ワイルド・カードを含むディレクトリ・エントリの検索では、Get File\$を続けて呼び出すことになりますが、一番最初の呼び出しでは、Func%を0に指定し、2度目以後は非0を指定します。2度目以後の呼び出しではFileName\$の文字列は意味を持ちません。

例

```
' ---/* ディレクトリ・エントリの検索 */---
'$INCLUDE: 'qb.bi'
'$INCLUDE: 'gen.bi'

DIM buf AS DtaType

INPUT "File Name ": f$

a$ = GetFile$(f$, Attr%, buf, 0)
DO WHILE a$ <> ""
  PRINT a$
  a$ = GetFile$("", Attr%, buf, 1)
LOOP
```

File Name ? *.hlp
QB45ADVR.HLP
QB45ENER.HLP
QB45QCK.HLP
QB45USER.HLP

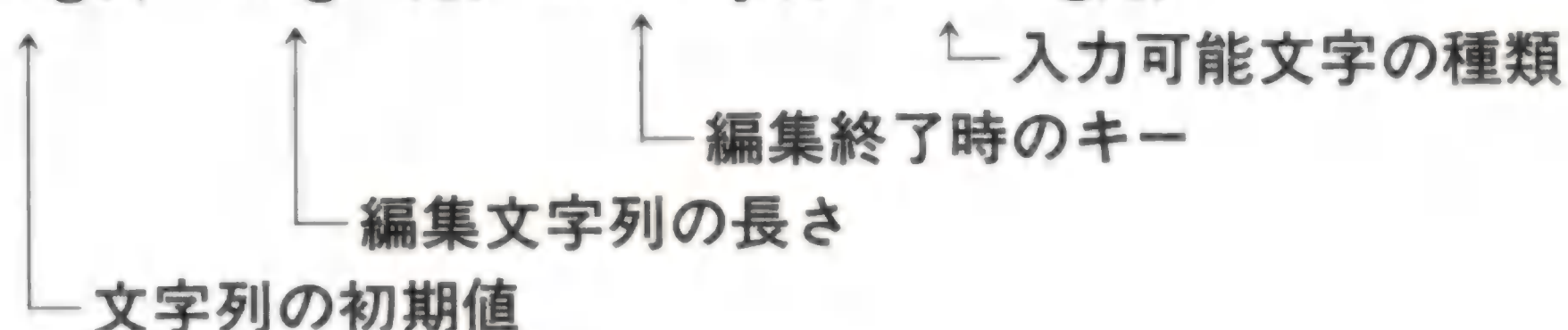
Edit\$

(文字列の編集)

関数

書 式

Edit\$(Arg\$, Length%, Lastkey\$, StrFlag%)



機 能

Arg\$の文字列が画面に表示され、その文字列を編集した結果が返されます。編集範囲は、Arg\$の文字列を含めて Length%で指定した長さです。[ESC], [↵]などのキーにより編集を終了しますが、このときの文字が Lastkey\$に返されます。

StrFlag%は、入力可能文字の種類で以下の値を指定します。

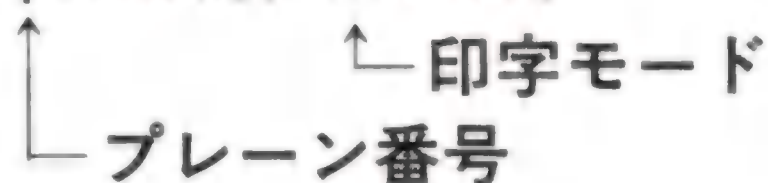
- < -1 半角
- = -1 FEP ON にして半角
- = 0 すべての文字
- = 1 全角
- > 1 FEP ON にして全角

HCOPY

(画面のハードコピー)

書 式

HCOPY plane%, mode%



機 能

テキスト画面とグラフィック画面のハードコピーをとります。

plane%に指定する値は、0～2のビットが印字するプレーンを示します。つまり第0ビット…Blueプレーン、第1ビット…Redプレーン、第2ビット…Greenプレーンとなります。

mode%が0ならノーマル、1なら縮小印字となります。

2 グラフィック・ライブラリ

GRAPH, QLB のライブラリについて説明します。このライブラリを使用するユーザプログラムでは, QB, BI と GRAPH, BI をインクルードしてください。

GROLL (グラフィック画面のスクロール)

書 式

GROLL x%, y%

↑ ↑
x 方向スクロールドット数 y 方向スクロールドット数

機 能

x%はx方向のスクロールドット数で8ドットの倍数を指定します。正の値なら右へ, 負なら左へスクロールします。

y%はy方向のスクロールドット数で1ドットの単位で指定します。正の値なら下へ, 負なら上へスクロールします。

KPUT (文字列のグラフィック画面への表示)

書 式

KPUT x%, y%, StrArg\$, fore%, back%

↑ ↑ ↑ ↑
表示位置(左上座標) 表示する文字列 フォアグラウンド色 バックグラウンド色

機 能

StrArg\$で示される文字列を, グラフィック画面の(x%, y%)位置に表示します。

LINEXOR (XOR モードでの直線)

書 式

LINEXOR x1%, y1%, x2%, y2%, plane%, style%

↑ ↑ ↑ ↑
始点 終点 描画プレーン ラインスタイル (0~&HFFFF)

機 能

描画面面との排他的論理和(XOR)をとり, その結果にしたがって線を引きます。描画は1プレーンのみで, 複数プレーンへは同時に描画できません。

plane% = 0 ... 青プレーン
 1 ... 緑プレーン
 2 ... 赤プレーン

3 マウスライブラリ

MOUSE. QLB のライブラリについて説明します。このライブラリを使用するユーザプログラムでは、QB. BI と MOUSE. BI をインクルードしてください。

MouseBorder (マウス移動範囲の設定)

書 式 MouseBorder $\underbrace{\text{row1\%, col1\%}}_{\text{左上座標}}, \underbrace{\text{row2\%, col2\%}}_{\text{右下座標}}$

機 能 マウス・カーソルの移動範囲を (row1%, col1%) – (row2%, col2%) に設定します。

MouseHide (マウスを非表示)

書 式 MouseHide

機 能 マウス・カーソルを表示しません。

MouseInit (マウスの初期化)

書 式 MouseInit

機 能 マウスの初期化を行います。マウスプロシージャの実行前に必ず一度実行しておく必要があります。
MouseInit 直後はマウスは非表示状態です。

MouseLocate (マウス位置の設定)

書 式 MouseLocate $\underbrace{\text{row\%, col\%}}_{\text{マウス位置の座標}}$

機 能 マウス・カーソルの位置を (row%, col%) に設定します。


```

' ---/* 2点間の直線を引く */---

'$INCLUDE: 'qb.bi'
'$INCLUDE: 'mouse.bi'

SCREEN 0: CLS
flag = 0: free = 1
Mouseinit
MouseLocate 200, 320
MouseShow

MousePoll y%, x%, Left%, Right%
DO WHILE Right% <> -1
  IF Left% = -1 AND free = 1 THEN
    IF flag = 0 THEN
      xs% = x%
      ys% = y%
      flag = 1
    ELSE
      MouseHide
      LINE (xs%, ys%)-(x%, y%), 1
      flag = 0
      MouseShow
    END IF
    free = 0
  END IF
  IF Left% = 0 THEN
    free = 1
  END IF
  MousePoll y%, x%, Left%, Right%
LOOP
MouseHide

```

始点/終点を表すフラグ
 右ボタンが押されるまで
 始点のときの処理
 終点のときの処理
 左ボタンが離されたときの処理

MousePush (マウス・ボタン ON 時の状態の取得)

書 式

MousePush button%, row%, col%, lbutton%, rbutton%
 ON ボタンの指定(0:左ボタン 1:右ボタン)
 マウスの現在座標
 左ボタンの状態
 右ボタンの状態

機 能

マウスの位置、ボタンの状態を取得します。得られたボタン状態が 0 ならボタン OFF、-1 ならボタン ON を意味します。

なお、button%には、最後にこのプロシージャが呼ばれたときから指定された(呼び出し時の button%で指定した)ボタンの押された回数が返されます。

MouseReady**(マウスが利用可能か調べる)****関数****書 式**

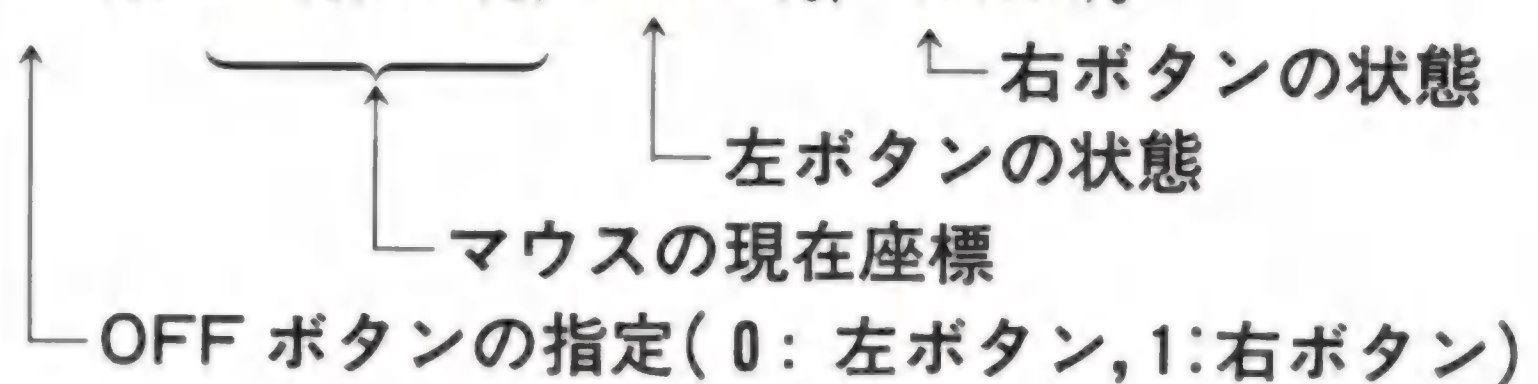
MouseReady

機 能

0 ... マウスが使える
-1 ... マウスは使えない

MouseRelease (マウス・ボタン OFF 時の状態の取得)**書 式**

MouseRelease button%, row%, col%, lbutton%, rbutton%

**機 能**

ボタンの状態を取得します。得られたボタン状態が 0 ならボタン OFF, -1 ならボタン ON を意味します。

なお, button%には, 最後にこのプロシージャが呼ばれたときから指定された(呼び出し時の button%で指定した)ボタンの離された回数が返されます。

MouseShow**(カーソルの表示)****書 式**

MouseShow

機 能

マウス・カーソルを表示します。

索引

●記号／英数字項目

\$DYNAMIC 144,310
\$INCLUDE 144,309
\$STATIC 144,310
8087/80287エミュレート 20,83

■ A

ABS 178
AND 124,271
ASC 178
ATN 178
AUTOEXEC.BAT 23

■ B

B 239
BC 17,80
BEEP 178
BF 239
BGM キュー 328
BLOAD 179
BSAVE 179
BYVAL 180

■ C

CALL 180
CALL ABSOLUTE 180
call by reference 139
call by value 140
CALL INT86OLD 181
CALL INT86XOLD 181
CALL INTERRUPT 169,182
CALL INTERRUPTX 182
CALLS 180
CDBL\$ 183
CDBL 183
CHAIN 184
CHDIR 184
CHR\$ 185
CINT 185
CIRCLE 186
CLEAR 187
CLNG 187
CLOSE 188
CLS 188

COLOR 188,331
COM 164,189
COMMAND\$ 67,190
COMMON 115,191
COMMON ブロック名 192
COMn 253
COMSPEC 18
CONS 253
CONST 104,193
context 339
COS 194
CSNG\$ 194,332
CSNG 194
CSRLIN 195
CVDMBF 196
CVI 196
CVL 196
CVS 196
CVSMBF 196

■ D

DATA 197
DATE\$ 197,198
DBL 200
DECLARE 198,199
DEF FN 199
DEF FN 関数 116,142
DEF SEG 168,200
DEFtyp 200
DGROUP セグメント 119
DIM 201
DO~LOOP 202
DOS 環境 169
DRAW 203
D/デバッグ 321

■ E

Easy メニュー 321
Edit\$ 344
END 205
ENVIRON\$ 205
ENVIRON 206
EOF 152,207
ERASE 207

ERDEV\$ 208
ERDEV 208
ERL 209
ERR 209
ERROR 210
EXIT 210,288
EXIT 文 132
EXP 211

■ F

FALSE 123
far ヒープ 287
far ヒープ領域 119
FIELD 211,273
FILEATTR 212
FILES 213
FIX 213
FOR~NEXT 214
FRE 214
FREEFILE 215
Full メニュー 321
FUNCTION 216

■ G

GEN. BI 342
GEN. QLB 342
GET 217,218
GetCommand\$ 342
GetFile\$ 342
GOSUB 219
GOTO 220
GOTO 文 132
GRAPH,QLB 345
GROLL 345

■ H/I/J

HCOPY 344
HELPMAKE 339
HEX\$ 220
H/ヘルプ 324
IEEE 形式 161,196,248
IF THEN ELSE 221
INKEY\$ 222
INP 223
INPUT # 225

INPUT 224
INPUT¥ 224
INPUT\$ 223
INSTR 226
INT 200,226
IOCTL\$ 227
IOCTL 227
IS 286
I/簡易ウォッチ 321
JIS\$ 227

■ K

KEXT\$ 228
KEY(n) 230
KEY 164,228
KILL 232
KINSTR 232
KLEN 233
KMID\$ 233,234
KPOS 234
KPUT 345
KTN\$ 234
KYBD 253

■ L

Last referenced point 264
LBOUND 235
LCASE\$ 236
LEARN 334
LEFT\$ 236
LEN 153,236,237
LET 237
LIB 18
LINE 237
LINE INPUT # 241
LINE INPUT 240
LINEXOR 345
LINK 85
LNG 200
LOC 241
LOCATE 242
LOCK 243
LOF 153,243
LOG 244
LP 264

LPOS 244
LPRINT 245
LPRINT USING 245
LPTn 253
LSET 245
LTRIM\$ 246

■ M

MID\$ 246,247
MKDIR 248
MKDMBF\$ 248
MKI\$ 247
MKL\$ 247
MKS\$ 247
MKSMBF\$ 248
MOUSE.BI 346
MouseBorder 346
MOUSE.COM 19
MouseHide 346
MouseInit 346
MouseLocate 346
MousePoll 347
MousePush 348
MOUSE.QLB 346
MouseReady 349
MouseRelease 349
MouseShow 349
MOUSE.SYS 19

■ N

NAME 248
NEW—CONF.SYS 318
NEW—VARS.BAT 318
NOT 124

■ O

OCT\$ 249
ON ERROR GOTO 163,249
ON event GOSUB 250
ON~GOSUB 251
ON~GOTO 251
ON PLAY 328
ON UEVENT 329
OPEN 252
OPEN COM 254

OPTION BASE 256
OR 125,271
OS シェル 77
OUT 257
O/オプション 322

■ P

PAINT 257
PALETTE 173,259
PALETTE USING 259
PATH 18
PCOPY 261
PEEK 262
PLAY 327,328
PLAY ON 329
PMAP 263
POINT 264
POKE 265
POS 265
PRESET 266,271
PRINT# 269
PRINT 266
PRINT.SYS 19
PRINT USING 267
PRINT USING\$ 269
PSET 270,271
PUT 271,273

■ Q

QB 16,28
QB.BI 342
QB45ADVR.HLP 335
QB45ENER.HLP 335
QB45QCK.HLP 335
QB45USER.HLP 335
QBE.EXE 312,326
QBGRPNEC 333
QB アドバイザ 312,324
QB チュートリアル 334
QB の起動オプション 29
QLB 341

■ R

RAM ディスク 18
RANDOMIZE 274

READ 274
REDIM 275
RegType 182
REM 275
RESET 275
RESTORE 276
RESUME 83,163,277
RETURN 277
RIGHT\$ 277
RMDIR 278
RND 278
RSET 279
RTRIM\$ 280
RUN 280

■ S

SADD 281
SCREEN 172,282,331,332
SCRN 253
SEEK # 285
SEEK 153,284
SEG 180
SELECT CASE 285
SETMEM 287
SETUP ユーティリティ 20
SGN 287
SHARE.EXE 243
SHARED 114,115,288
SHARED 属性 191
SHELL 288
SIN 289
SLEEP 330
SNG 200
SOUND 329
SPACE\$ 290
SPC 290
SQR 290
STATIC 291
STEP 238
STOP 291
STR\$ 292
STR 200
STRING\$ 292
STRING 108
SUB 292

SWAP 293
SWITCH コマンド 17
SYSTEM 293

■ T

TAB 294
TAN 295
TIMES 295
TIMER 164,296
TIMER ON 296
TMP 18
TROFF 297
TRON 297
TRUE 123
TYPE 112,297

■ U/V/W/X

UBOUND 298
UCASE\$ 299
UEVENT ON 330
UNLOCK 243,299
VAL 299
VARPTR\$ 301
VARPTR 300
VARSEG 300
VIEW 301
VIEW PRINT 155,303
VRAM 179,262
WAIT 303
WHILE~WEND 304
WIDTH 305,332
WINDOW 306
WRITE # 307
WRITE 307
XOR 125,271

●和文項目

■ ア

アクティブ・ウィンドウ 38
アクティブ画面 37
アクティブ・ページ 174
値による呼び出し 140
アドバイザディスク 316

アナログ機 259
アン・ドゥ 59
イベント・トラッピング 164,189
入れ子 210
色番号 173,189
インクルード 309
インクルード・ファイル 48,144
インタプリタ 61
インデント 54
インプリシット 200
インプリシット宣言 106
ウィンドウ 306
ウィンドウの分割 37
ウォッチウィンドウ 37
ウォッチ機能 70
ウォッチ式 70
ウォッチ・ポイント 71
エディタ 50
エラーコード 209
エラー処理 163
エラー・トラッピング 163
エラー・トラッピングルーチン 249
演算子 121
応答ファイル 87
オート・インデント 54
オープンモード 151,252
オフセット値 168
オペランド 99
オリジナルスクリーン座標 171,301
オンラインヘルプ 75,324

■カ

カーソル形状 242
階層ディレクトリ 156
カスタマイズ 60
型宣言文字 105
型変換 120
画面のハードコピー 344
画面幅 305
画面モード 172
仮引数 139
カレント・ディレクトリ 156,184
環境の設定 17
環境変数 206
環境文字列 205

関係演算子 122
漢字 158
漢字連続コード 159
関数プロシージャ 136,216
偽 123
キー・トラッピング 230
キー配置 50
キーワード 324
記憶クラス 117
記号定数 104,193
基本データ型 107
行識別子 99
行番号 99
共用変数 191
近代的流れ制御構造 127
クイック・ライブラリ 64,65
グラフィック 171
グラフィック座標 171
グラフィック・モード 172
グラフィック・ライブラリ 345
クリック 31
クリップボード 55
グローバル変数 114
構造化プログラミング 127
後判定反復 127,131,202
固定小数点書式 266
固定長文字列 108
コマンド・ライン 67
コマンド・ライン引数 190
コメント 99,275
混合演算 120
コンソール入出力 155
コンパイラ 62
コンパイル・オプション 166

■サ

サーチ 56
サーチパス 323
再帰 142
再帰プロシージャ 140
サブディレクトリ 156,248
サブメニュー 32
サブモジュール 42,147
サブルーチン 133,219
サブルーチン・プロシージャ 135,292

算術演算子 122
参照による呼び出し 139
シーケンシャル・ファイル 150, 225, 241
時間 167
式 121
字下げ 54
四捨五入 185
辞書順 125
システムコール 169, 181
実行可能ファイル 62
実行制御 146
実引数 139
自動構文チェック 53
自動変数 118
シフト JIS 159
従来型 IF 文 128
16進数 103
条件判断 221
ショート・カット・キー 36
所定回反復 127, 129, 214
真 123
数値演算コ・プロセッサ 326
数値演算コ・プロセッサ8087 162
数値処理 160
スーパーインポーズ・モード 172
スクリーン座標 171
スクリーンモード 282
スクロール 345
スコープ 114
スタックサイズ 187
スタック領域 117
スタンドアロン型 20
スタンドアロン・ライブラリ 64
ステートメント 99
スリープ 330
制御構造 127
整数型 107
静的配列 144, 193, 310
静的変数 118, 291
静的領域 117
セグメント 168
セグメント・アドレス 200
セグメント値 168, 300
セットアップディスク 313
宣言 135

相対指定 238
前判定反復 127, 130, 202, 304
属性番号 173, 188
ソフトウェア割り込み 169, 181, 182

■タ

ダイアログ・ボックス 39, 57
代替数値演算 20, 83
タイトル・バー 38
タイマ・トラッピング 167
タイルパターン 258
ダイレクトモード 72
ダイレクトモード・ウィンドウ 37
タグ 112
タブ・サイズ 59
単純 IF 文 128
単精度実数型 107
チェイン 184
通信ポート 189
通用範囲 114
定義 135
定数 103
ディスク・ファイル 156
ディスプレイ・ページ 174, 282
ディップスイッチ 17
ディレクトリ・エントリ 342
データ型 102
データ型の変換 161
テキスト・ビューポート 303
テキスト・モード 172
デジタル機 259
デバイス 157
デバイス・ファイル 154
デバッグ 69
動的配列 82, 119, 144, 193, 275, 310
ドキュメント・ファイル 47
独立型 64
とび越し 132
トラッピング 163
トラッピングルーチン 165
トレース・オフ 297
トレース・オン 297
トレース機能 69

■ナ

内部コード形式 196
名前付き COMMON 192
日本語 FEP 18
ぬりつぶし 257

■ハ

倍精度実数型 107
排他的論理和 125
倍長整数型 107
ハイパーリンク 325,336
配列 109
配列添字の下限值 235
配列添字の上限値 298
パスの設定 18
パス名 151
8進数 103
バックグラウンド・カラー 188
パレット 259
パレット・モード 173,189,259,282
判断 127
汎用ライブラリ 342
引数 134
引数渡し 138
ヒストリ 73
ビット演算 124
ビット反転 124
否定 124
ビューウィンドウ 37
ビューポート 155,301,306
表示色 173
ファイル・エンド 152
ファイル現在位置 241,285
ファイル・サイズ 243
ファイル処理 149
ファイル属性 212
ファイル・タイプ名 151
ファイルのオープン 252
ファイルバッファ 211,279
ファイル番号 151,252
フィールド 111
フォアグラウンド・カラー 188
複数条件判断 129,285
物理アドレス 168

物理座標 263
浮動小数点書式 266
部分範囲指定 110
プライマリー・ファイル名 151
プリンタ・ドライバ 19
プリンタ・ヘッド 244
ブレーク・ポイント 71
プレース・マーカ 58
プレーン 179
プログラムディスク 314
プロシージャ 133
ブロック 221
ブロック IF 文 128
ブロック・コピー 55
ブロック・ムーブ 55
ベーシック・コンパイラ 80
ヘルプ 76
ヘルプファイル 335,338
編集 50
編集コマンド 51
変数 104
変数管理 148
変数の共有 115
ポート 223,257,303

■マ

マイクロソフト・バイナリ形式 162,196,248
マウス・ドライバ 19
マウス入力方式 31
マウスライブラリ 346
マクロコマンド 203
マスク 124
マルチ・ステートメント 99,221
未使用ファイル番号 215
メインメニュー 33
メインモジュール 42
メタコマンド 144,309
メニュー操作 30
メモリ・サイズ 215
メモリ操作 168
メモリの設定 17
メンバ 111,297
メンバの参照 113
文字型 107
文字セット 98

モジュール 42
文字列演算 125
文字列処理 158
文字列の検索 226
戻り値 137

■ヤ/ラ/ワ

ユーザ定義 230
ユーザライブラリ 341
優先順位 121
予約語 100
ライトアクセスモード 252
ライブラリディスク 315
ライブラリ・マネージャ 91
ラインスタイル 239
ラジアン 289
ラベル 58,99
乱数 278
乱数列の初期化 274
ランタイム分離型 20,64
ランダム・ファイル 150,217,273
リードアクセスモード 252
リテラル定数 103
リプレイス 56
リンカ 85
ルート・ディレクトリ 156
レコード型 111,297
レコード型変数 153
レコード番号 150,152
接続 127
ローカル変数 114
論理演算子 123
論理座標 263
論理積 124
論理和 125
ワールド座標 171,306

河西朝雄(かさい あさお)

昭和49年	山梨大学工学部電子工学科卒
現在	長野県岡谷工業高等学校情報技術科教諭
	第一種情報処理技術者
著者	「Cプログラミング技法」, 「Cプログラミングノート」, 「Cプログラミング入門」, 「MS-DOSシステムコールハンドブック」, 「TURBO PASCALハンディ・マニュアル」(ナツメ社)「構造化BASIC」, 「MS-DOS実用マクロアセンブラ」, 「TURBO C初級プログラミング上・下」, 「最新はじめてのBASIC」, 「最新はじめてのTURBO C」(技術評論社) など

Ver.4.2～4.5

QuickBASIC初級プログラミング入門(上)

平成2年4月30日 初版 第1刷発行

平成5年12月25日 初版 第9刷発行

著者 河西朝雄

発行者 片岡 巖

発行所 株式会社技術評論社

東京都新宿区愛住町8番地8

電話 03(3225)2300 営業部

03(3225)3293 編集部

印刷／製本 加藤文明社

定価はカバーに表示してあります

本書の一部または全部を著作権法の定める
範囲を超え、無断で複写、複製、転載、テ
ープ化、ファイルに落とすことを禁じます。

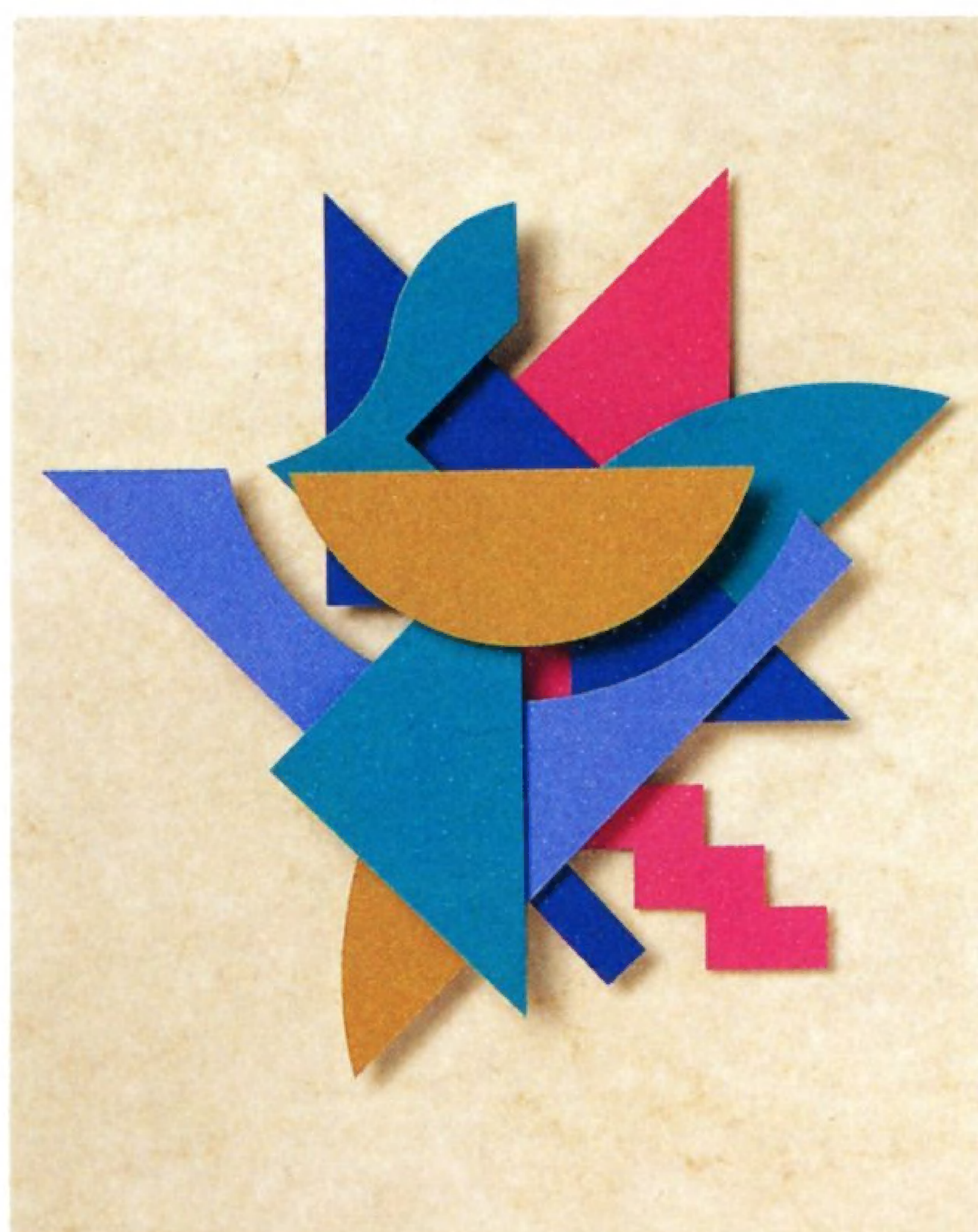
©1990 河西朝雄

ISBN4-87408-353-6 C3055

Printed in Japan

M-cord 210041

Ver.4.2~4.5
Quick BASIC
初級プログラミング入門[上]
河西朝雄



ISBN4-87408-353-6 C3055 P2100E 定価2100円(本体2039円)